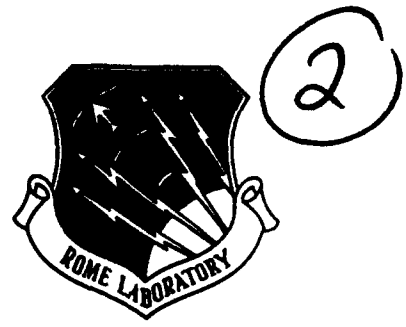


RL-TR-92-9, Vol II (of three)
Final Technical Report
January 1992

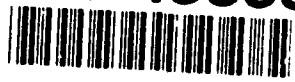
ASSURED SERVICE CONCEPTS AND MODELS Security in Distributed Systems

Secure Computing Technology Corporation

Sponsored by
Strategic Defense Initiative Office

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Strategic Defense Initiative Office or the U.S. Government.



Rome Laboratory
Air Force Systems Command
Griffiss Air Force Base, NY 13441-5700

This report has been reviewed by the Rome Laboratory Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RL-TR-92-9, Vol II (of three) has been reviewed and is approved for publication.

APPROVED: Joseph W. Frank

JOSEPH W. FRANK
Project Engineer

FOR THE COMMANDER: John A. Graniero

JOHN A. GRANIERO
Chief Scientist
Command, Control & Communications Directorate

If your address has changed or if you wish to be removed from the Rome Laboratory mailing list, or if the addressee is no longer employed by your organization, please notify RL (C3AB), Griffiss AFB NY 13441-5700. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

ASSURED SERVICE CONCEPTS AND MODELS
Summary

J.T. Haigh
R.C. O'Brien
W.T. Wood
T.G. Fine
M.J. Endrizzi
S. Yalamanchili

Contractor: Secure Computing Technology Corporation
Contract Number: F30602-90-C-0025
Effective Date of Contract: 23 February 1990
Contract Expiration Date: 22 February 1991
Short Title of Work: Assured Service Concepts and Models
Period of Work Covered: Feb 90 - Feb 91

Principal Investigator: Tom Haigh
Phone: (612) 482-7400

RL Project Engineer: Joseph W. Frank
Phone: (315) 330-2925

This research was supported by the Strategic Defense Initiative Office of the Department of Defense and was monitored by Joseph W. Frank, RL (C3AB) and Emilie J. Siarkiewicz, RL (C3AB) Griffiss AFB NY 13441-5700 under Contract F30602-90-C-0025.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	January 1992	Final Feb 90 - Feb 91

4. TITLE AND SUBTITLE
ASSURED SERVICE CONCEPTS AND MODELS
Security in Distributed Systems

5. FUNDING NUMBERS
C - F30602-90-C-0025
PE - 63223C
PR - 3109
TA - 01
WU - 01

6. AUTHOR(S)
J. T. Haigh, R. C. O'Brien, W. T. Wood, T. G. Fine,
M. J. Endrizzi, S. Yalamanchili

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Secure Computing Technology Corporation
1210 West County Road, East
Suite 100
Arden Hills MN 55112

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Rome Laboratory (C3AB)
Griffiss AFB NY 13441-5700

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
RL-TR-92-9, Vol II (of three)

11. SUPPLEMENTARY NOTES
Rome Laboratory Project Engineers: Joseph W. Frank/C3AB (315) 330-2925
Emilie J. Siarkiewicz/C3AB (315) 330-3241

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

This report describes the work performed under the Assured Service Concepts and Models (ASCM) contract. The report is organized as follows. Volume I is a summary of all of the work done in the ASCM project. Volume II describes the various security policies that were developed on the contract. Volume III describes the availability policies that were developed on the contract and the approaches that were developed for identifying trade-offs between secrecy and availability. Volume III also contains the findings of the formalism study.

14. SUBJECT TERMS
Computer Security, Assured Service, Availability, Distributed Systems

15. NUMBER OF PAGES
88

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	U/L

CONTENTS

SECTION 1

INTRODUCTION

This volume is a final report on research into security policies and models for distributed C^2 systems. The research has been performed under the Assured Service Concepts and Models (ASCM) project. The goals of the ASCM project are to:

a. Develop security policies and models for distributed C^2 systems.

b. Develop availability policies and models for distributed C^2 systems.

c. Analyze trade-offs between security and availability.

d. Analyze a real-system with respect to the policies and models developed on the project.

This report addresses the first of these tasks.

1.1 PURPOSE

Considering the extensive research that has been conducted into formalizing the notion of security in computer systems, it is natural to ask why more research is necessary. The simplest answer is that the policies that have been developed to date are not sufficient to address the concerns present in distributed C^2 systems.

To begin with, it is not clear that the security policies that have been developed for nondeterministic systems are adequate. Since there usually is some degree of nondeterminism in a distributed computing system, it is necessary to deal with nondeterministic systems before considering distributed systems. The nondeterminism is caused both by the concurrency that is present whenever the system has multiple CPUs and by uncertainties in the communications medium.

Second, the broadcast protocols used to address the uncertainties in the communications medium can lead to additional security concerns. For example, broadcast protocols often rely on a sender being able to observe an acknowledgement from a receiver to determine whether a broadcast message is received. In the case where the receiver is allowed to receive messages from the sender but not allowed to send messages to the sender, this might introduce a security flaw. The receiver might be able to send messages to the sender indirectly through acknowledgements.

Finally, there are features required for C^2 systems that are not necessarily required for distributed systems. The features that will be considered in this report are those of an adaptive nature. These features allow a system to adapt its processing as it evolves. The specific adaptive features that will be considered in this report are:

a. Reclassification of Data — It is not uncommon for the classification of a piece of data to change with time. For example, plans for a battle are probably more sensitive before the battle than after the battle. It is also possible for data to become more sensitive.

b. System Reconfiguration — As new nodes are added to a system or nodes in a system fail, the system is reconfigured.

c. Change of Operational Mode — The rules defining the correct operation of the system can change over time.

Before attempting to formalize the concept of security in a distributed C^2 system, we provide an example system in Section **2**. Using the intuition provided by the example system, we define in Section **3** what we believe to be the control objectives for the security policy. Next, in Section **4** we provide a brief survey of policies that have been developed to date. In addition to describing existing policies, we discuss deficiencies present in these policies and suggestions as to how some of these deficiencies can be addressed. Since we view adaptive security policies as addressing exceptions to standard security policies, we feel that it is important to address deficiencies in these standard security policies before attempting to address adaptive security policies. Then, in Section **5** we discuss adaptive security policies and security issues related to broadcast protocols. In this section we propose a new security policy, *TH-Guardedness* as an "adaptive security policy." In Section **6** we discuss relationships that hold between various security policies. In particular, we examine the so-called composability issue. Finally, in Section **7** we summarize the report and pose questions that require further research.

SECTION 2

EXAMPLE C^2 ADAPTIVE SECURITY POLICY

Adaptive Security Policies are ushering in the next generation of security policies. Currently, security policies such as the DOD Bell-LaPadula Policy[25] and various Discretionary Access Control policies[8] typify a static form of security policy. Once such a policy is set, it remains in force until the system administrator or a user decides to alter the policy. In contrast, adaptive security policies are dynamic in nature. An adaptive security policy modifies access decisions based upon the properties of the system to which it is applied and the environment in which the system operates. If these factors change, the policy will adapt to these changes in a pre-defined manner.

Examples of adaptive security policies abound. In the financial markets it is illegal for corporate insiders to leak certain information until a formal announcement is made. In government, politicians are not required to disclose honoraria until they exceed $100,000.00. In defense, war plans remain secret until their disclosure no longer threatens the mission. In each of these examples, information is kept secret until a specific property of the system or environment changes: a formal announcement, exceeding the honorarium limit, reduced mission threat. At that point a different policy on the flow of information is implemented.

One of the purposes of this report is to present models of adaptive security policies. The models will be based on a hypothetical example of an Air Force C^2 system with well understood security requirements. This section will present a top-level description of this system and its security requirements.

2.1 OVERVIEW

This example was derived from the Military Airlift Command (MAC)[26] and models a C^2 system responsible for transportation. It is not intended to perfectly model MAC but instead to describe the relevant features common to most C^2 systems.

C^2 systems have the responsibility to ensure the timely initiation of the planning, scheduling and execution of missions. It is not the system's job to load the cargo or turn the wrenches, but to guarantee these tasks are done, and, if necessary, to acquire and allocate resources needed to complete them.

C^2 systems have unique security requirements. C^2 security requirements typically change as the system enters different modes of operations. For example, in peace mode, force capability information is typically unclassified while in war mode it may become top secret. It is difficult to address such "exceptions" in a conventional static security policy. Thus, adaptive security policies must be developed to cover these changes in operational modes.

This section will describe the concept of operations, the organization, the information flows, and the system modes for the C^2 example.

2.1.1 Concept of Operations

The concept of operations addresses the command and information flow through the C^2 system during the course of a mission (Figure 2-1). The three major participants in the C^2 system are the initiator of the transport request, Headquarters (HQ) which is responsible for global management, and Field Units (FU) which are responsible for local management in the field.

The concept of operations revolves around moving cargo from source to destination. C^2 responsibilities begin when HQ receives a request to transport cargo. At this time the initiator will list all the requirements associated with the request such as cargo type, times, sensitivity of planning information, embarkation and disembarkation points, etc. HQ will respond with a package of alternative courses of action to service the transportation request. It is possible that the initiator will iteratively ask "What if several of the requirements are altered to enhance the service?" HQ will respond with a possibly different course of action package.

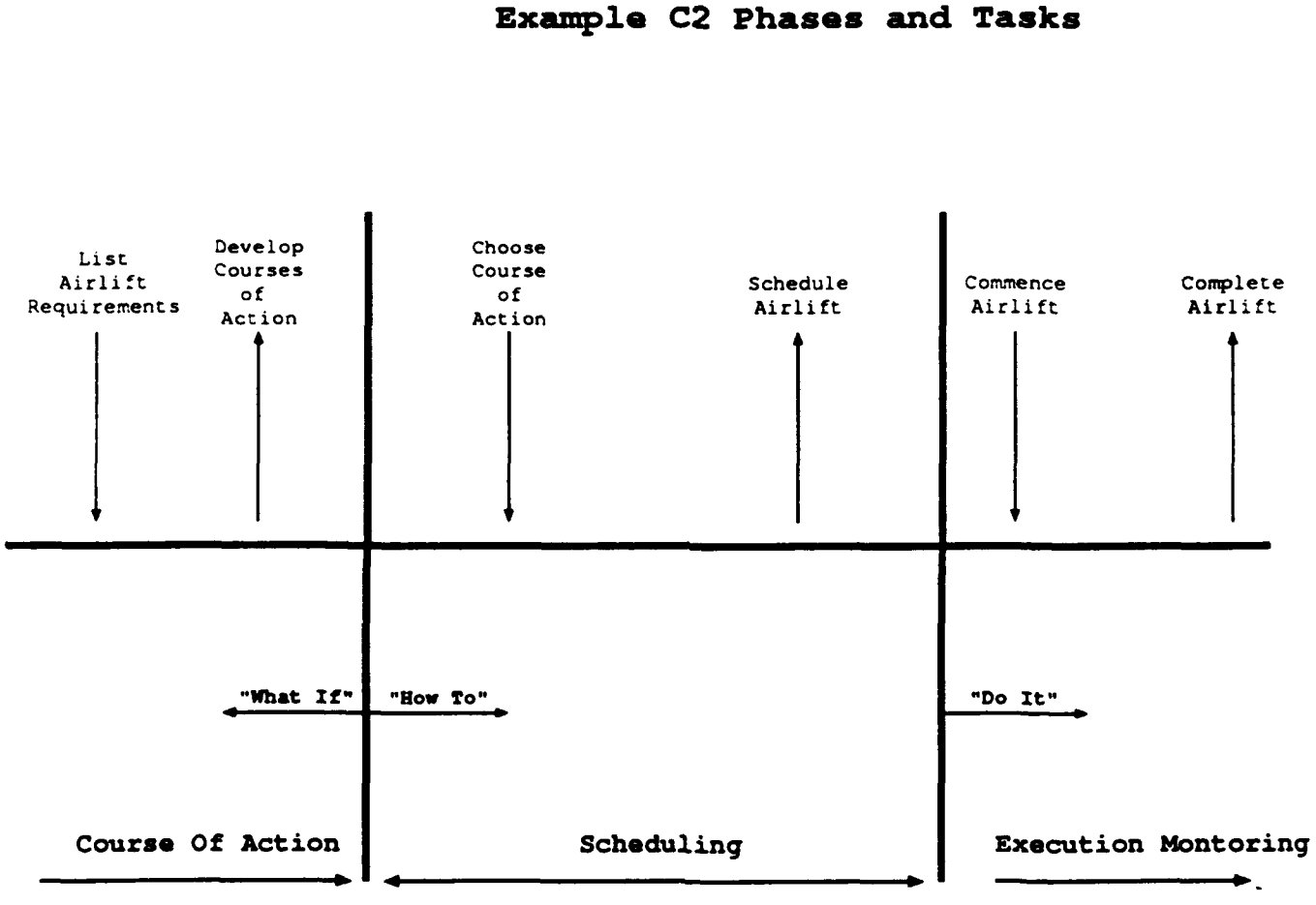


Figure 2-1. C^2 Phases

When the initiator acquires sufficient information to make a decision, a specific course of action is chosen for HQ to plan and implement. HQ is then responsible for scheduling the resources to satisfy

the course of action. HQ will respond with a transport schedule that will commence sometime in the future.

At the appropriate time, HQ will commence the transport. The C^2 system is now responsible for monitoring the execution of the transport. When the transport is complete, HQ will notify the initiator that the request for transport has been fulfilled.

2.1.2 Organization

The C^2 organizational structure for this example is composed of a headquarters(HQ) and field units(FU) (Figure 2-2). HQ is responsible for accepting transport requests and interacting with one or more FUs to accomplish the task. The major functions of this C^2 organization are Operations, Transportation and Logistics.

Operations is responsible for gathering information from its supporting groups and then planning, scheduling and monitoring the mission. Transportation is responsible for allocating sufficient resources to transport the cargo and ensuring the cargo arrives at its destination. Logistics is responsible for allocating maintenance and support resources and ensuring that the system maintains a level of capability needed to complete the mission.

These three major functions are further subdivided into specific roles (Figure 2-3). Roles are job positions with specific responsibilities. One example of a role from Figure 2-3 is OP.COA.HQ. OP.COA.HQ has the responsibility of operations and planning during the course of action phase at the Headquarters level. This role is commonly called a mission planner.

The interaction between these roles is depicted by the lines in Figure 2-2. Each role communicates directly with its support group. Although interactions may be composed of several lower level iterative interactions, this example will only focus on the top-level system interaction.

2.1.3 Information Flows

The information flows that result from the interactions among roles are depicted in Figures 2-4 through 2-5. The nodes of these graphs represent the roles described in Figure 2-3. The boxes represent the type of information that is allowed to flow between two roles. The arrows describe what information flows between roles. For example, in Figure 2-5, Weather is a type of information that flows from FU Operations to HQ Operations during the Course of Action phase.

The general flow is initiated by the Joint Deployment Community (JDC). HQ will take JDC information and then pass it down to the FUs. The FUs process the information and then return additional information to HQ. This cycle is repeated for each mission task as depicted in Figure 2-1.

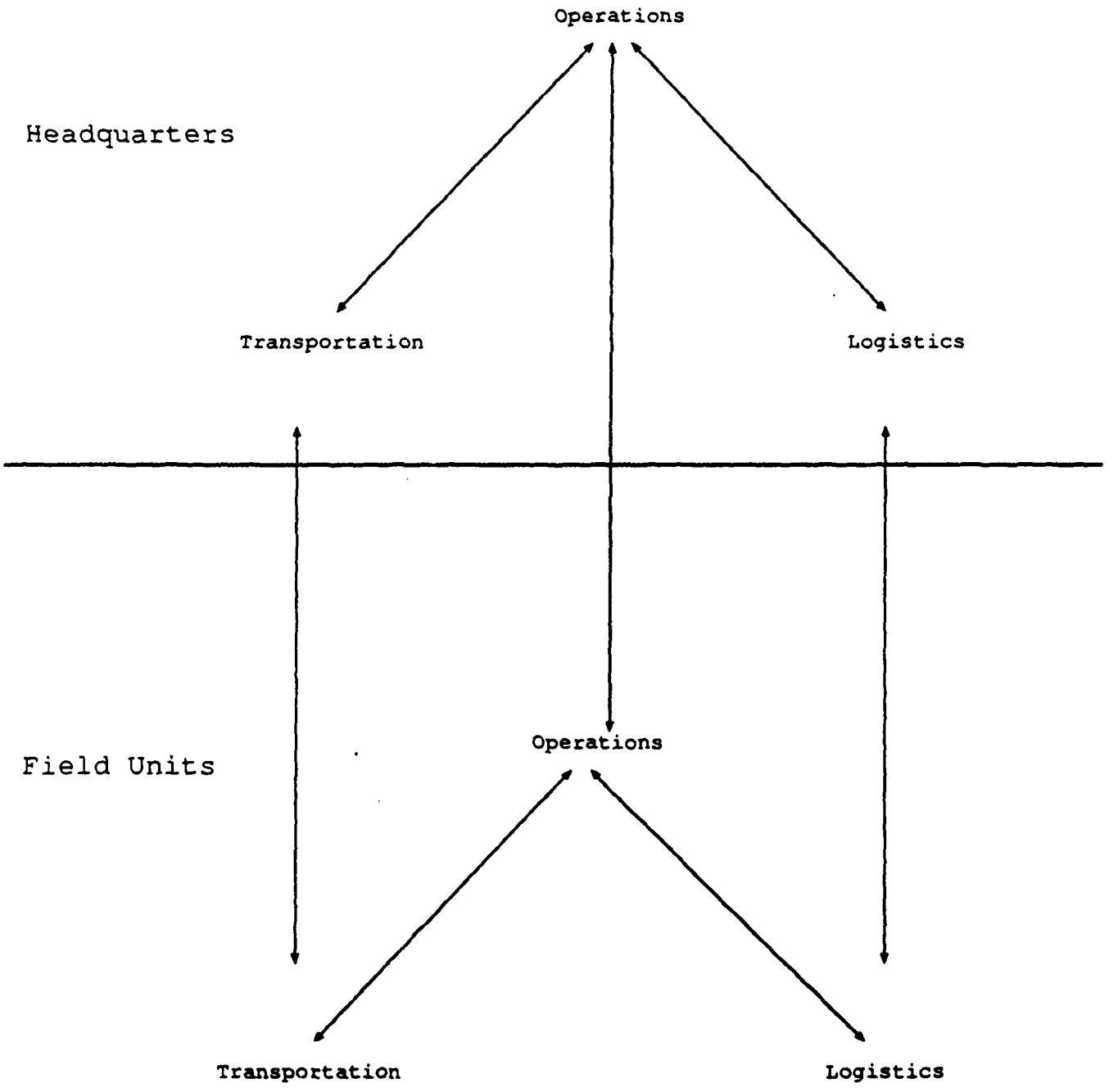


Figure 2-2. C^2 Structure and Functions

C2 Example Role Definition

	Course of Action (COA)	Schedule (S)	Execution Monitoring (M)
HeadQuarters (HQ)	OP.COA.HQ LO.COA.HQ TR.COA.HQ	OP.S.HQ LO.S.HQ TR.S.HQ	OP.M.HQ LO.M.HQ TR.M.HQ
Field Units (FU)	OP.COA.FU LO.COA.FU TR.COA.FU	OP.S.FU LO.S.FU TR.S.FU	OP.M.FU LO.M.FU TR.M.FU

OP = Operations and Planning
LO = Logistics and Maintenance
TR = Transportation and Airlift

Figure 2-3. C^2 Roles

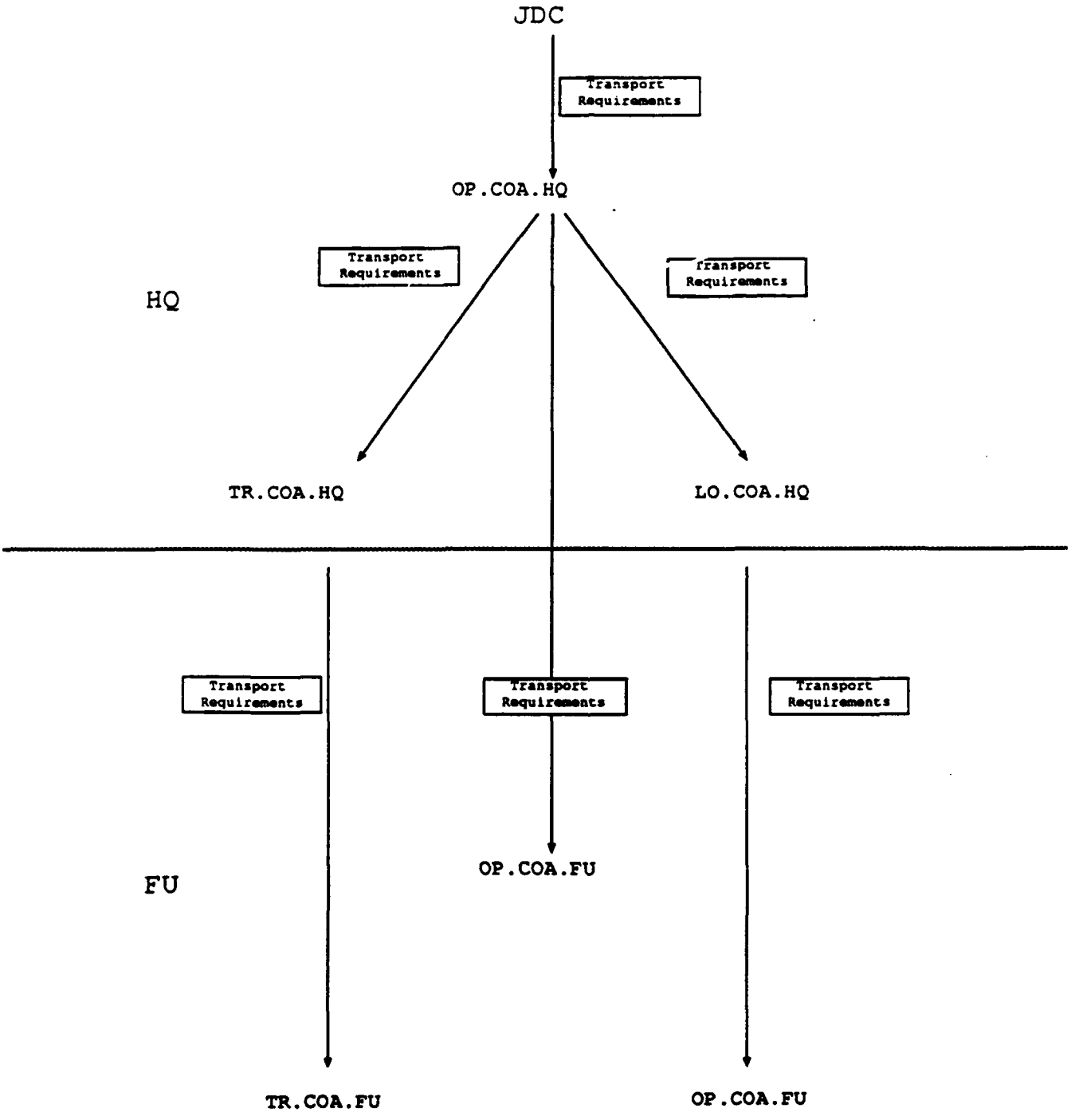


Figure 2-4. Course of Action - List Transportation Requirements

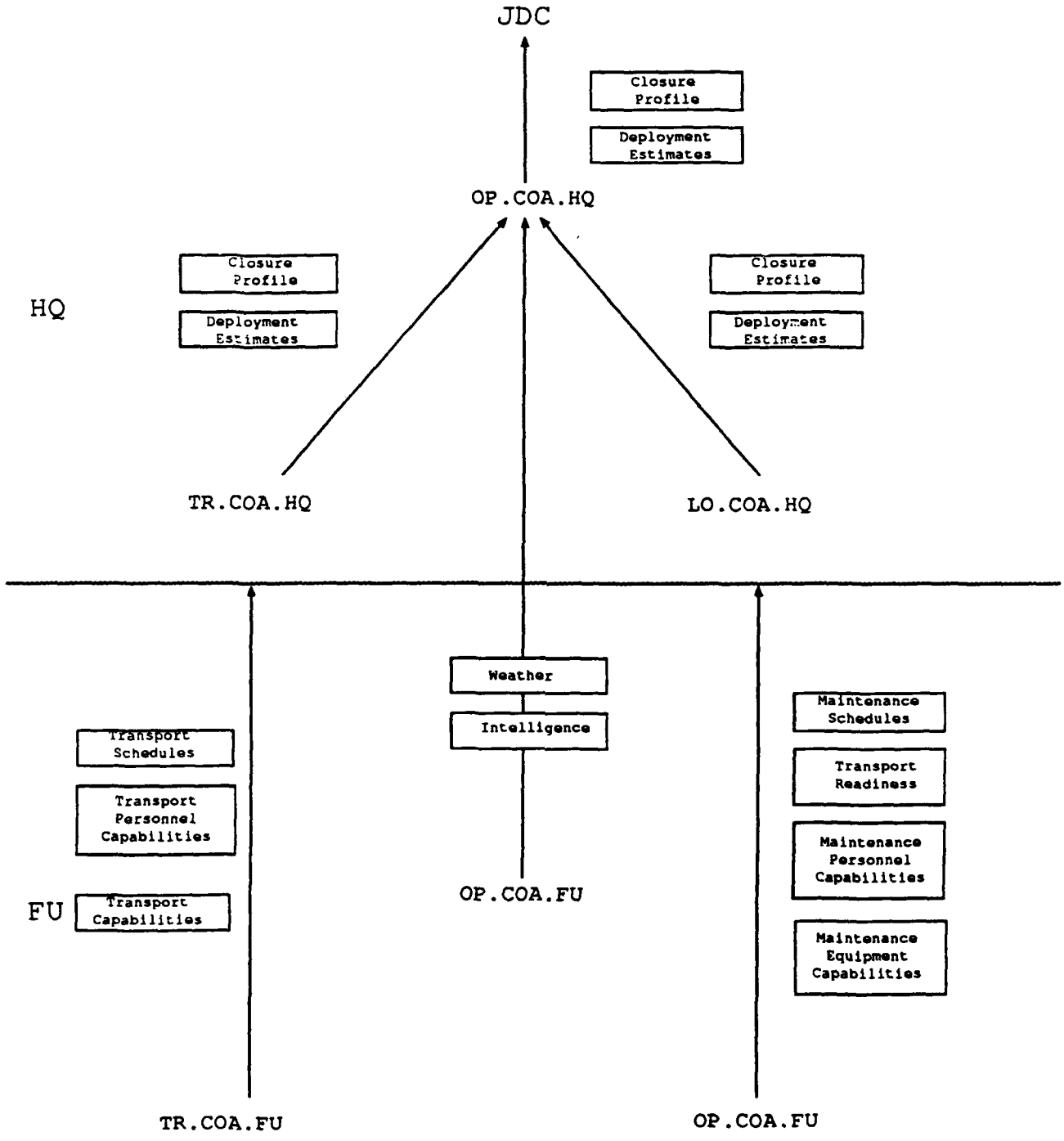


Figure 2-5. Course of Action - Develop Courses of Action

The first information flow is initiated by the JDC (Figure 2-4). JDC provides a list of transport requirements to HQ. These requirements contain information such as type of cargo, transport locations and transport time. HQ passes this information to the FUs that will participate in the transport. Each FU will determine the local resources that must be made available to service this request. Each FU function will return one or more alternative plans to its parent HQ function (Figure 2-4). This information contains forecasts of capabilities, weather, intelligence, availabilities, etc. With this information HQ will prepare a package of course of action plans and deliver them to the JDC(Figure 2-5).

Figures 2-4 and 2-5 describe information flows for the Course of Action phase of the mission. Information flows for the remaining two phases (scheduling, execution) exist but contain redundant information that does not add to the content of this report. Therefore, they will not be discussed at this time.

2.1.4 System Modes

This example C^2 system can operate in three modes: training, war, and peace. Training mode permits users to experiment with different scenarios without fear of corrupting real information because data is duplicated or fabricated. Peace mode is the normal operating mode of the system. In peace mode most missions are scheduled long in advance and short notice "critical" missions are the exception. In war mode, short notice "critical" missions are dictated by the situation in the battlefield and secrecy of information is a greater concern. In peace and war modes, long range planning must co-exist with the realities of short notice missions.

2.2 ADAPTIVE SECURITY POLICY

Since adaptive security policies are based upon properties of the system, if there are many properties the policy may become very complex. To avoid such complexities, the adaptive security policy for this C^2 example will only contain the following components:

Role Flow Rule - Restricts flows between roles.

Mode Rule - Restricts flows based on modes.

The Role Flow Rule refers to the information flows described in section **2.1.3**. These information flows subtly describe one type of adaptive security. During each phase of the mission only specific information flows are permitted. For example, the Transport Requirements are not available during the execution monitoring phase. More formally:

> **Role Flow Rule**: The restriction of information flow is based upon the types of data and the roles in the system. The policy rules specific to the system will explicitly state what types of data are permitted to flow between roles. Any flows outside of the scope of these policy rules are in violation of the policy.

Note that the Role Flow Rule does not explicitly restrict information flow based on mission phase. This restriction is implicitly resident in the definition of a role. A role is defined in terms of mission phases. Using roles, the Role Flow Rule indirectly restricts the flow of information between mission phases.

A second way in which the information flows are sensitive to their environment is related to the operational modes of the system. Depending on the mode of the system, each information flow is assigned a specific hierarchical security level. More formally:

> **Mode Rule**: For all data, a hierarchical security level for each system mode will be assigned. The level will reflect the sensitivity of the data for a given system mode. Users must be cleared to the level of the data in order to access the data regardless of the role they are performing or the mode of the system.

Figures 2-6 to 2-7 depict the security levels assigned to information flows per system mode, per mission phase. Training mode is the simplest to implement as all information is classified at *sensitive but unclassified.* Most peace mode information flows are classified at *sensitive but unclassified*, but there are *secret* flows for capability information and *top secret* flows for handling intelligence related information. Most war mode information is classified at *secret* with capability and intelligence information classified at *top secret* .

The most interesting aspect of this policy occurs when the system changes modes. Data will suddenly be reclassified, thus introducing a period of uncertainty. For example, a database management process could be working on unclassified information in peace mode when suddenly the system goes into war mode, thereby revoking the access that the process has to now top secret information.

While this example has described a policy that adapts to the mode of the system, no policy has been given for moderating the changes between system modes. This will be explored further in Section **5**.

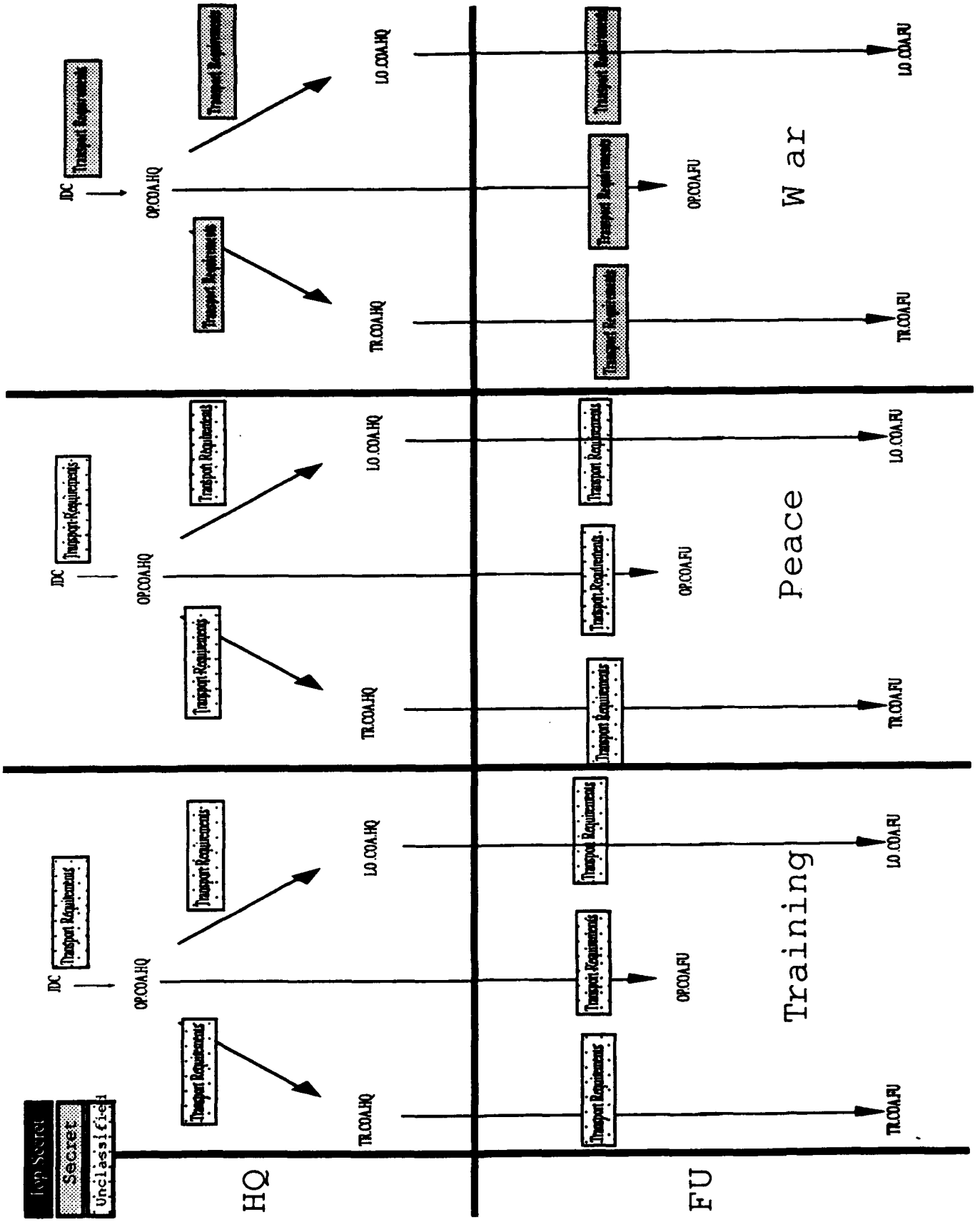


Figure 2-6. Course of Action - List Transportation Requirements Policy

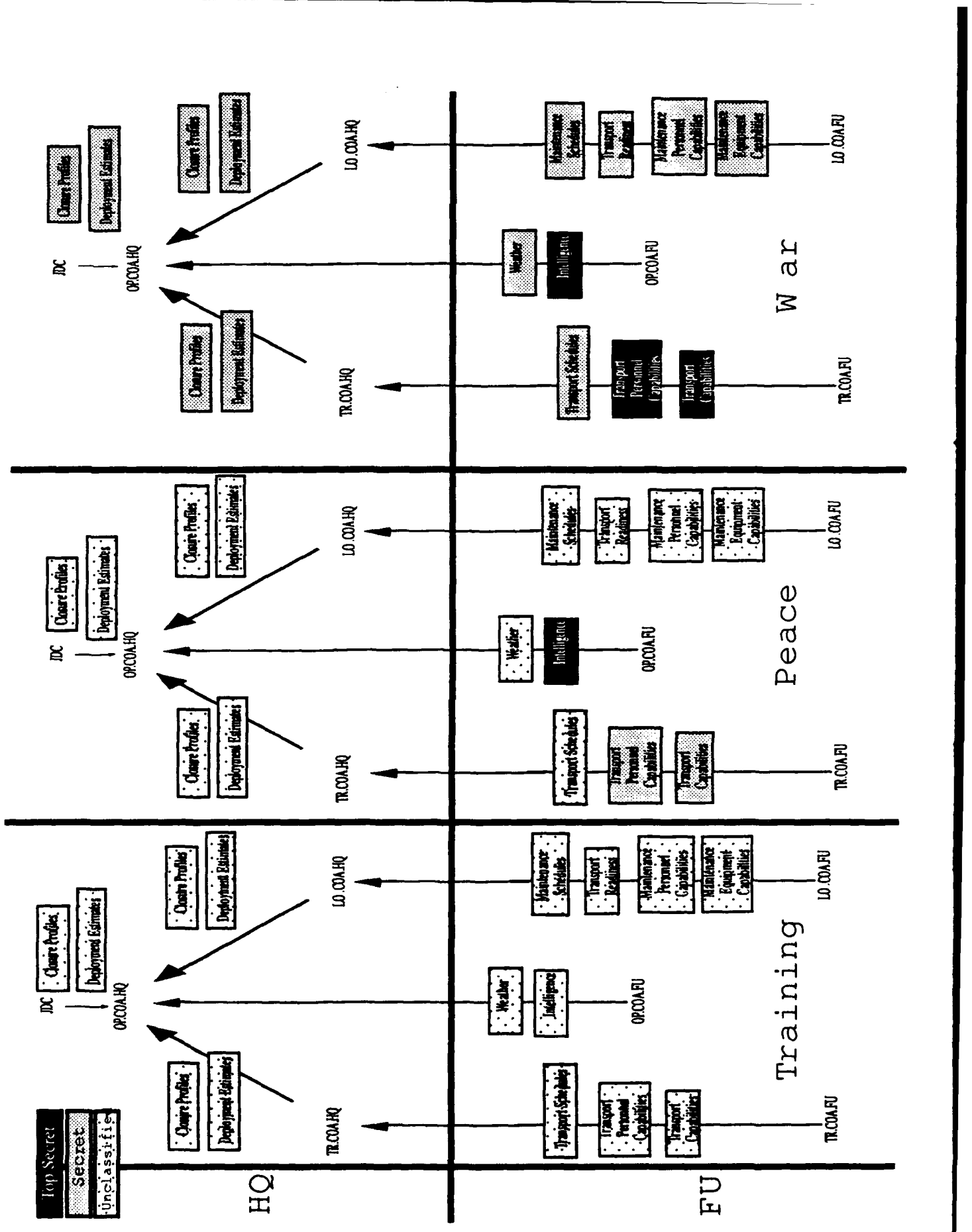


Figure 2-7. Course of Action - Develop Courses of Action Policy

SECTION 3

CONTROL OBJECTIVES FOR A C^2 SYSTEM

In the previous section we presented an example of a C^2 system. In this section, we will use the intuition provided by the example system to help formalize the objectives of security policies for C^2 systems.

The specific control objectives for a C^2 system are closely related to its mission. However, for this study we are interested in security control objectives, which tend to be fairly generic. The computer security problem is generally defined as the problem of protecting against unauthorized:

- Disclosure of data,
- Modification of data, and
- Denial of system service.

In this discussion we include executable object code as data.

In this report we address the first two problems. The corresponding control objectives are the secrecy control objective and the integrity control objective. We will address the problem of protection against unauthorized denial of service in a future report, where we will formulate an availability control objective. However, insofar as availability requires data and program integrity, the problem is addressed by the integrity control objective.

In this section we describe each of the control objectives. These control objectives are stated in terms of the authorizations of users to acquire information or modify data. We distinguish between data and information in the following manner. Information is associated with human users of the system. It is encoded in the system as data. Users extract information from data using whatever context and auxiliary information they might have. Thus, certain classes of covert channels are the result of private encodings of information in data. At the receiving end of the covert channel, either the user extracts the information from the data or, more likely, a subject running on the user's behalf decodes the data and presents the user with a different set of data in which the information is encoded in a more conventional fashion.

Before stating the control objectives, we present a description of the sort of environment in which we envision the system is operating. This description is a generalization of the example in Section 2.

3.1 THE OPERATING ENVIRONMENT

We assume that the system under examination is a military, distributed, C^2 system. Here we discuss the ramifications of each modifier, military, distributed, and C^2 , separately.

Since we are studying a military system, we expect that there is a partially ordered set (poset) of security levels associated with the system. Every dataset stored on the system is associated with a security level that reflects the sensitivity of the information that the data is intended to represent. Similarly, each user of the system has an associated security level that indicates the level of the most sensitive information that the user is authorized to acquire. We do not assume that security levels form a lattice. Thus, it is not necessarily true that each set of security levels has associated high water mark (least upper bound) and low water mark (greatest lower bound) security levels. In particular, there may not be a unique System High security level. We will, however, assume the existence of a unique System Low security level.

The standard DoD security lattice, characterized as the product of the lattice of hierarchical security levels and the lattice of subsets of the set of all categories, is a special case in which the security poset has a special structure. Thus, the standard DoD lattice is one choice for the security poset of the system. On the other hand, we address a much broader class of systems in which highly sensitive information can be compartmentalized so that no single user may acquire access to it all.

Since the system is distributed, we expect that functional differences among the nodes will have an impact on the control objectives, and we expect security policies that are specific to the nodes. For instance, in the example, the control objectives will differ for the different command echelons. Moreover, the precise formulation of the security policy at a node will depend upon the functions performed at the node. This will have an impact on the overall policy, which must be compatible with both the control objectives for the overall system and with the control objectives and policies of the different nodes.

We must also expect that at times the mode of operation will vary among the nodes. In the example, most nodes might be operating normally, while one node is off the network for maintenance and a training exercise is underway on another node. While this is the case, communications among the nodes must be restricted, and it may be necessary to transfer some of the functions normally performed by the nodes currently in maintenance and training modes to other nodes. This is one reason for replication of data at the various nodes.

Because a C^2 system has a clearly defined mission, there are certain functions that must be performed, and many users can be identified with clearly defined functions, or roles, on the system. These users must perform specific, mode dependent actions on specific classes of data, and they need not perform any other actions. Moreover, it is neither desirable not necessary for these users to write and install their own programs. Their set of permissible actions can be encapsulated in a well-defined set of programs developed for their use. Thus, the accesses of many users to code and data can be restricted based on their roles and the current mode of operation of the system.

On the other hand, for these users to properly perform their duties, they must have access to correct information. This means that certain datasets on the system must be faithful representations of the information that they are intended to represent. To ensure this, modifications of these datasets must be very carefully controlled. Since these datasets may not necessarily represent sensitive information, they may not be associated with high security levels. To distinguish data that must be protected against compromise from data that must be protected against unauthorized modification, we call the latter *critical data*, reserving the term *sensitive data* for the former.

These features:

- The control of access to information, and therefore data, by security level,
- Heterogenity of objectives, policy, and mode of operation across the network, and
- Separation of users and data by role, or function,

are the distinguishing characteristics of the systems that we are examining in this study. In the next two subsections we present appropriate secrecy and integrity control objectives for such systems.

3.2 THE SECRECY CONTROL OBJECTIVE

The secrecy control objective can be stated as follows:

> *It shall not be possible for an individual to use the C^2 system to acquire sensitive information unless the individual is authorized to know the information.*

Information is sensitive either because it is classified or because the user responsible for the information determines for some other reason that the data is sensitive. In the former case, the sensitivity of the information is measured by the associated security level. In the latter case, the responsible user indicates the sensitivity of the information by restricting the accesses of various individuals and groups of individuals to the object(s) in which the corresponding data is stored. Because of the broad set of possible options, the discretionary access component of the security policy is not addressed further in this section.

Normally, the acquisition of information by a user occurs when a subject acting on behalf of the user directly observes data in which the information is encoded. However, information can also be acquired indirectly via the interactions of a chain of subjects at various security levels, acting on behalf of one or more users. This is how covert channels are exploited. One or more subjects manipulate the system in such a manner that the effects visible at a lower or incomparable level can be used to encode information that is decoded by the actions of one or more subjects at the target level. It has been demonstrated [14] that a security policy prohibiting direct acquisition of high level information by a user does not necessarily prevent indirect acquisition of the information by the user. MLS noninterference policies [27] prohibit indirect as well as direct acquisition of higher level information by users.

To achieve the secrecy control objective, we use a conditional MLS noninterference (CNI) policy that is a generalization of the one developed for the LOCK system [31]. The policy is conditional because downward interferences are inherent in certain required functions for a C^2 system, such as the reclassification of data and mode changes. The CNI policy is described in Section **5.1**. We then supplement this policy with specific policies for each of the exceptions in Section **5.2**.

Correct enforcement of both the CNI policy and the policies for the exceptions rely on the integrity of the security database used by the trusted computing base (TCB) to enforce the policy. This integrity is a consequence of the integrity control object that we present in the next section.

3.3 THE INTEGRITY CONTROL OBJECTIVE

Integrity is a more complex concept than secrecy. The secrecy of data is independent of the semantics of the data, but, as Clark and Wilson have demonstrated, this is not true for integrity [6]. Data integrity can be considered a combined measure of the degree to which the data accurately reflects the external entity which it is intended to model and of the degree to which it is consistent with a set of internal constraints. Generally, the internal constraints reflect structural requirements determined by the semantics of the data. They differ from external constraints in that the consistency between the data and internal constraints can be checked without further reference to the state of the external entity. For both classes of constraints, the semantics of the data are critical in determining the integrity of the data, and any operation on the data must reflect those semantics if the integrity of the data is to be preserved or enhanced.

Since semantics are application dependent, any notion of integrity depends upon the intended application for the data and the associated operations on the data. Thus, the maintenance of data integrity requires a partnership between the application programs that are intended to operate on the data and the underlying access control mechanisms of the system. Each application program must operate correctly in the sense that, viewed as a function from data sets to data sets, it will, if properly applied, always transform data in its domain into data with the desired integrity properties. The underlying system access control mechanisms must restrict accesses so that only users who can be trusted to correctly apply the function will have execute access to the program and so that the program can only accept as input, data from the appropriate domain.

We now incorporate these comments into the following integrity control objective:

> *If an individual can modify critical data on a C^2 system, then the modification can only occur as the result of an action which the individual is authorized to perform on the data and which has been certified to preserve the integrity of the data.*

The integrity control objective has been influenced by the work of Clark and Wilson [6]. Later, in Section **6.2** we elaborate on the relationship between the integrity control objective and Clark-Wilson integrity policies. We also show how LOCK type enforcement [4] can be used to describe such policies.

SECTION 4

FOUNDATIONS OF SECURITY

Because adaptive security policies are the composition of existing policies, a review of existing policies is warranted. The two general categories of security policies are access control policies and information flow policies. As we are more concerned with the secrecy control objective in this report than the integrity control objective, the majority of this section deals with the secrecy control objective. After describing access control policies, we will discuss how information flow policies address flaws encountered when using access control policies in the context of the secrecy control objective. We will then discuss attempts to generalize information flow policies to nondeterministic and stochastic system models. After pointing out flaws in these attempts, we will suggest our own information flow policies for these systems. The next section will then build upon these policies to provide adaptive security policies.

4.1 ACCESS CONTROL POLICIES

Access control policies moderate the interaction between subjects (users/processes) and objects (files). The simplest model of an access control policy is represented by a two-dimensional access control matrix (Fig 4-1). The components of this model are:

- Subjects
- Objects
- Access Permissions
- Access Matrix

One dimension of an access control matrix represents subjects, S_n. Subjects are active entities which at a high-level of abstraction represent users and at a low-level represent processes or jobs acting on the behalf of a user. All subjects are labeled with ids and are assumed to be memoryless.

The other dimension of an access control matrix represents objects, O_n. Objects are passive entities that are operated on by subjects. Objects are repositories of protected information and labeled with ids. Note that a computer program is stored in an object up to the point where it is executed by a subject. When stored as a computer program it is labeled and referenced through an object id, and when executed it is labeled and referenced through a subject id.

Access permissions state the permissible operations on an object. In the basic model these permissions include read, write, execute, delete, and append. These three components(subjects, objects, and access permissions) are combined in an access matrix to state an access control policy. The policy rules are located in the the intersection between S_i and O_j. That is, the $(i,j)^{th}$ entry in the matrix indicates what operations S_i is permitted to perform on the data stored in O_j.

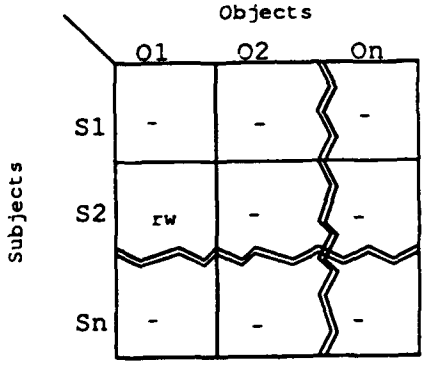


Figure 4-1. Access Matrix

We say that a system satisfies the policy if subjects are only permitted to access objects in ways that are consistent with the access control matrix.

In order to apply access control policies to systems, it is necessary to interpret the access control matrix in the context of the system, to determine what it means for a subject to access an object in the system, and to determine the manner in which the access control matrix can be altered. By varying these factors, the general access control policy can be instantiated in several ways. In the remainder of this section will discuss several of these instantiations: DAC, Bell-LaPadula, Clark-Wilson, Type Enforcement, and Chinese Wall. We then describe a Generalize Framework for Access Control (GFAC) [23] that can be used to combine the various access control policies. In all of these policies, we will see that the focus is on representing the access control matrix and controlling modifications to the access control matrix. Surprisingly, the question of what it means to access an object is ignored in access control policies. For example, the question of what it means to read or write an object is typically ignored in these policies. They simply require that whenever a subject has access to read or write an object, the security policy allows that subject to read or write the object. There is not, however, an explicit requirement that a subject needs to have access to read or write an object in order to actually read or write the object.

4.1.1 Discretionary Access Control(DAC)

Early computer systems had no forms of internal access control. Access control consisted of a computer operator mounting the appropriate tape for the appropriate job. With the advent of on-line disk storage the responsibility of access control shifted from operators to users, hence the term Discretionary Access Control (DAC) [1].

DAC is typically used to control the access that a subject has to objects based on the user on whose behalf the subject is operating. As there are often sets of users to which it is desirable to grant the same accesses, it is quite common to provide the capability to specify *groups* of users and the accesses they have to each object. This is merely a convenience, though, and provides no additional functionality.

As computer systems mature, a profusion of users and objects are added to the access matrix complicating management and security analysis. It is also very inefficient to store a large access matrix that is sparsely populated. To address these weaknesses, row and column based representations of the access matrix were formulated.

4.1.1.1 Row Based DAC Policies — Row based DAC policies include capabilities, profiles and passwords. These DAC policies are considered row-based representation of the access matrix because associated with each subject is a list of objects that it has access to. There is no global access matrix, all access information is local to the subject (Figure 4-2) [12].

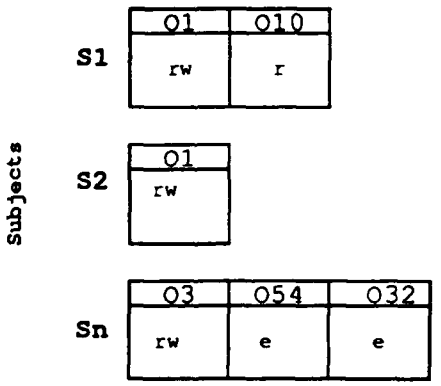


Figure 4-2. Row Based DAC

[1] *Access control specified and maintained by the user and not the system (hence, at the users discretion).*

The most common of these representations are capabilities [18]. A subject obtains a capability by creating and owning the object, or through another subject that passes the capability. A capability contains information about the owner subject, access rights, and a crypto checksum. The crypto checksum ensures that subjects do not maliciously forge capabilities or manipulate access rights.

A subject obtains access to an object by searching through its capability list for the capability or "ticket" to the object. The subject then presents the capability and desired mode of operation to the system. The system verifies the capability and permits/denies the subject access to the object. Additional policies moderating the creation, passing, and modification of capabilities have been formulated [13][18][33].

Profiles [8] are very similar to capabilities with the exception that users do not present a profile to the system when requesting access to a resource. Instead the system is responsible for scanning a subject's profile searching for authorization to the resource.

Passwords [8] are also similar to capabilities, they both require users to present a "ticket" in order to access the object. Objects are password protected. When a subject requires access to an object, the subject must present a password(the "ticket") to the object. Some systems have a different password for every access mode to an object.

In all of these methods we see that the access control matrix is divided into structures that define the objects to which each subject/user has access and the corresponding accesses permitted. In general, modification of the pieces of the access control matrix is at the discretion of subjects/users.

4.1.1.2 Column Based DAC — Row-based DAC policies complicate the revocation of privileges and access review because of duplicated object information. Object information is duplicated because each subject maintains a separate copy of the access privileges to a specific object. If the system decides to revoke access to an object, it must search through the capability lists of all the subjects in the system for the specific object.

Column based DAC is a representation of the access control matrix that centralizes control over the object(Figure 4-3). The system performs access reviews by searching through an object's access list which is stored locally with the object. Column based DAC policies include Access Control Lists (ACL)s and protection bits.

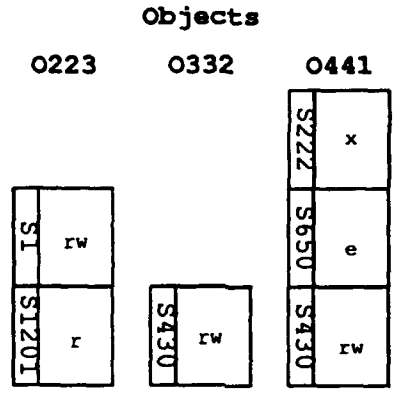


Figure 4-3. Column Based DAC

An ACL[8] is a list of users(groups) that are allowed/denied access to an object. Associated with each user in an ACL is the access privileges for that user. When a subject acting on behalf of a given user requests access to an object, the system searches through the object's ACL for the users access rights. If access rights are present, the system permits the subject to access the object. With current technology, ACLs are the best way to implement an access control matrix by column. ACLs are the only mechanism that allows users to explicitly allow and deny access to an object.

Protection bits[8] are a coarser form of ACL most commonly found on Unix systems. Instead of explicitly listing the users that are permitted access to an object, protection bits only specify access permissions granted to the owner, a group or all users (world privileges). Protection bits do not have the capability to explicitly deny access to a specific user as do ACLs.

In all of these methods we see that the access control matrix is divided into structures that define the subjects/users that can access each object and the corresponding accesses permitted. In general, modification of the pieces of the access control matrix is at the discretion of subjects/users.

4.1.2 Bell-LaPadula

A common security flaw common to all DAC mechanisms is readily exploited by Trojan horses. A Trojan horse is a program that whiffffffle performing a useful task surreptitiously steals or modifies information. For example, suppose that user A has an object that he does not want user B to read, user B wants to read the object, and that user A attempts to protect the object by using a DAC mechanism to prevent subjects operating on the behalf of user B from reading the object. Now, suppose that user B writes a program that while appearing to perform some useful task surreptitiously copies the object into a second object that user B can read. If user B runs the program, then the DAC mechanism will prevent the data from being copied since the subject executing the program would be operating on user B's behalf. On the other hand, if user A runs the program, then the subject executing the program will be operating on user A's behalf and the data will be copied. So, user B can obtain information in violation of the DAC policy by tricking A into running the program and reading the copied data. The cause of this problem is that the

policy does not address the transitive nature of information flow. If user A can read X and write Y and user B can read Y, then one must recognize the possibility that X can be transmitted to B through Y.

A similar problem occurs when the permitted accesses can be manipulated by subjects. Continuing with our above example, user B's program might change the permissions associated with the object so that user B has access. Then, after tricking user A into running the program, user B can read the object at will.

In order to thwart such attacks, the DOD mandatory access control policy formalized by Bell and LaPadula [2]:

- Enforces a mandatory policy rather than a discretionary policy.
- Addresses the transitive nature of information flow.

Bell-LaPadula consists of 4 components:

- Hierarchical Security Levels.
- Non-Hierarchical Categories.
- Clearances and Classifications by Access Classes.
- Access Rules.

Hierarchical levels form a linearly ordered lattice based on dominance. Each level instantiates a common set of non-hierarchical categories. Dominance for categories is determined by a lattice of subsets ordered by the inclusion relation [7]. An access class consists of a level-category pair.

Every subject/object has a related access class. For a subject, the level indicates the level of trust associated with the subject and the category set indicates the set of topics about with the subject needs to know. It is tacitly assumed that the clearance level and "need to know" of the user associated with the subject are greater than the clearance level and category set of the subject. For an object, the level indicates the maximum sensitivity of data that can be stored in the object and the category set indicates the set of topics that can be related to data stored in the object[12].

It is common practice to define a partial ordering on access classes by:

$(l_1, c_1) \preceq (l_2, c_2)$ if and only if:

- l_1 is less than or equal to l_2 with respect to the ordering on levels.
- $c_1 \subseteq c_2$

and to refer to state policies in terms of access classes rather than hierarchical levels and categories. It is also common practice to then ignore the distinction between access classes and levels and simply consider systems with partially ordered security levels rather than systems with linearly ordered security levels and partially ordered category sets. We will follow this practice and refer to access classes as levels in the remainder of this report.

A subject's accesses to an object are restricted by the following access rules: [12][24]:

- Simple Security Property - A subject may have read access to an object only if the object's level is less than or equal to that of the subject.
- *-property - A subject may have write access to an object only if the level of the object is greater than or equal to that of the subject.

In a system constrained by these rules, information only flows upwards in level. The simple security property permits high level subjects to read information at the same or lower level than itself (pull information upwards). The *-property permits low level subjects to write information at the same or higher level than itself (write information upwards).

So, we see that the Bell-LaPadula policy implicitly defines the access control matrix by the Simple Security Property and the *-property. It is a mandatory policy, rather than a discretionary policy, in that it is assumed that neither the levels of subjects and objects nor the partial ordering on security levels can be modified at the discretion of users. It addresses the transitive nature of information flow by using a transitive ordering on the security levels. More explicitly, if A can read X and write Y and B can read Y, then:

- Since A can read X, A's level dominates that of X.
- Since A can write Y, Y's level dominates that of A.
- Since B can read Y, B's level dominates that of Y.
- Then, the transitivity of the ordering on levels implies that B's level dominates that of X.

In other words, whenever the policy allows B to indirectly obtain X through Y, it allows B to obtain X directly. Conversely, if B cannot obtain X directly, then B cannot obtain X indirectly through Y. This is in contrast to DAC in which B can obtain X indirectly even if A has used the DAC mechanism to prohibit B from directly accessing X.

4.1.3 Integrity Policies

The Bell-LaPadula model of multilevel security supports the first goal of all security policies, protecting the confidentiality of the data. Soon after the Bell-LaPadula model was developed people began to develop integrity policies that described the permissible modifications to information. Two integrity policies are discussed in this section: Biba, and Clark-Wilson.

Biba integrity [3] is useful for users assigning a course granularity of integrity to objects. Biba's integrity model retains Bell-LaPadula's hierarchical levels but inverts the rules for reading and writing. Low level subjects (possibly malicious) are not permitted to write up to higher level objects thus protecting the integrity of high level data. High level subjects are not permitted read down from lower level objects thus protecting the integrity of high level subjects and objects. Note that the set of levels used in stating a Biba integrity policy for a system is typically different than

the set of levels used in stating a Bell-LaPadula policy for the same system. To avoid confusion, the former is commonly referred to as the set of integrity levels while the latter is referred to as the set of security levels. So, although it might seem that the Biba integrity policy conflicts with the Bell-LaPadula security policy, there is no conflict since they deal with two totally different sets of levels.

Clark and Wilson expanded the focus of security policies by defining an integrity policy for the commercial world [6]. In the commercial world, security is based on the roles that people play; accounts receivable, accounts payable, shipping, etc. Associated with these roles are well defined duties, constrained inputs and outputs, and interactions with other roles. For example, a clerk in accounts receivable only deals with billing customers while a clerk in accounts payable only deals with paying a firm's bills. Rarely do these two roles interact.

Clark-Wilson integrity formalizes roles with two main concepts; well formed transactions (WFT), and separation of duties. WFTs are operations that have been certified to preserve or introduce integrity properties of data. Clark and Wilson use WFTs to state integrity policies where certain integrity preserving operations are required to transform information from one format to another. For example, consider the role of an auditor. The auditor is trusted to digest a firm's financial information, authenticate its validity and produce a financial bill of health. Thus, programs for accomplishing these tasks would comprise a natural set of WFTs to provide the auditor role.

The separation of duties concept relies on the assumption that if the completion of a task requires the actions of several individuals, then information can be stolen or modified through the performance of the task only through collusion of some of the individuals. Separation of duties increases the risk of detection and decreases the chances of success for potential attackers. An example of separation of duties is the opening of a bank vault where two independent bank officers are required to insert their keys and type in their security codes. Separation of duties may further require that these actions occur in a specific order.

The policy presented in section **2.1.3** is an example of a Clark-Wilson integrity policy. Figure 4-4 (derived from figure 2-5) is a good example of WFTs and separation of duties. The roles OP.COA.FU, LO.COA.FU and TR.COA.FU are certified to perform specific actions (WFTs) on specific datasets (in Clark-Wilson terminology, Constrained Data Items). OP.COA.FU has been certified to gather weather information and send it to OP.COA.HQ, TR.COA.FU has been certified to build a transport schedule and send it to TR.COA.HQ, and LO.COA.FU has been certified to build a maintenance schedule and send it to LO.COA.HQ. The FU roles are not permitted to interact (separation of duties) thwarting any malicious collusions intended to corrupt information.

This portion of Clark-Wilson is an access control policy because it moderates the interactions between subjects and objects. The access control matrix is defined by associating with each constrained data item the set of WFTs that can access it along with the permitted accesses. Thus, the access control matrix in a Clark-Wilson policy is similar to the access control matrix in a column based DAC policy. There is a significant difference between a Clark-Wilson policy and a column based DAC policy in that a Clark-Wilson policy includes rules governing the manner in which the access control matrix can be modified that make it a mandatory policy rather than a discretionary policy. In addition, Clark-Wilson includes aspects that go beyond access control policies.

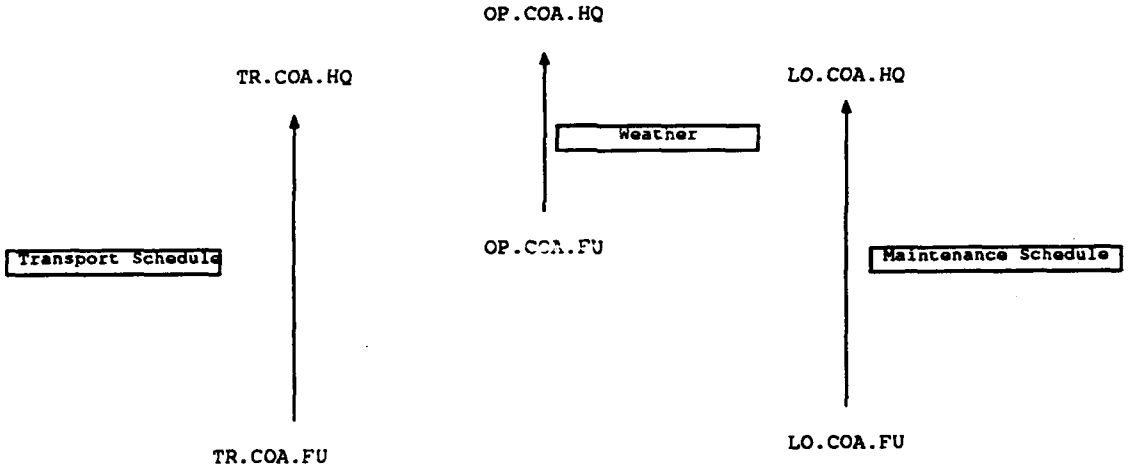


Figure 4-4. C^2 Example of WFT and Separation of Duties

4.1.4 Type Enforcement

Type enforcement is a policy that can be used as a foundation for a variety of integrity policies, including Clark-Wilson policies. The basic concept behind type enforcement is a mandatory version of an access control matrix and restrictions on the classes of subjects that can be associated with a given user (see [19] and Section **4.1**). The difference between type enforcement and the basic access control matrix described in **4.1**, is the granularity of control over subjects and objects. Access control matrices require users to specify access permissions on a per subject-object basis. In type enforcement, subjects are grouped into equivalence classes known as domains and objects are grouped into equivalence classes knows as types (Figure 4-5). Access decisions are made on the granularity of domains and types instead of subjects and objects. Access decisions are then applied to the subjects and objects of those domains and types.

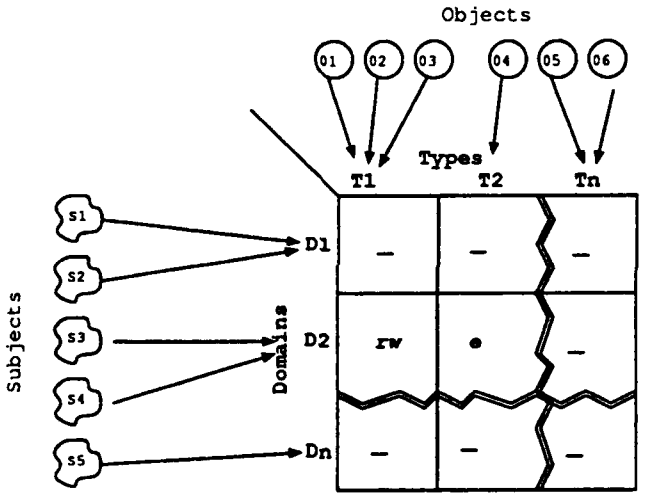


Figure 4-5. Type Enforcement

For the example, Figure 4-5 shows how subjects S_1 through S_5 are grouped into domains D_1 through D_n. A similar type of grouping applies to objects O_1 through O_6 into types T_1 through T_n.

The policy stated by this matrix is that S_3 and S_4 are permitted to read and write objects O_1, O_2, and O_3 as well as execute O_4.

The access control matrix specifying the accesses subjects in each domain are permitted to objects of each type is called the domain definition table (DDT). A second table, the role authorization table (RAT) associates each user with the set of domains in which the user can have subjects operating.

Type enforcement is extremely flexible. By proper configuration of the DDT and RAT, it is possible to state a wide variety of properties. Additionally, research is being performed on application specific policies such as Role-Based Application Policies [29].

4.1.5 Chinese Wall

All the access control policies discussed to this point (DAC, Bell-LaPadula, Clark-Wilson) describe policies that are somewhat static. The users or the system sets the access permissions and they remain in force until explicitly changed. The Chinese Wall policy [5] popularized a new generation of security policies with a dynamic flavor. Similar dynamic policies are found in the database security world (LOCK Data Views or LDV) and are used for aggregation and inference control [9].

These policies are dynamic because access permissions are not explicitly set by the system but instead are dependent on what information a subject has accessed in the past. When a subject makes a request to access new information, the system consults with its security database to ensure that new information does not conflict with any old information the subject may still have in its memory. If there is no conflict, access is permitted. If there is a conflict, access is denied.

An example of where such a policy might be used is in controlling the access that an accountant that consults for large corporations has to various pieces of information. In order to maintain personal integrity, the accountant should not consult with two competing corporations at the same time for fear of releasing sensitive information. The accountant should only work with the second competitor after the conflicting information becomes old or purged.

4.1.6 Generalized Framework for Access Control

The Chinese Wall policy opens up a Pandora's box for access control policies. Previously, access control policies used very simple rules to moderate the interactions between subjects and objects. For example:

- subject X is permitted read access to object Y, or
- a subject at Secret can write to objects at Top Secret.

Implementing the Chinese Wall policy requires that these rules become more complex and cover additional attributes beyond ids and security levels. First, an information history must be kept for all subjects. This history will maintain a list of the information the subject had access to in the past. Second, access control authorities must be able to cleanly identify what information generates conflicts and translate these conflicts into rules. The rules must then look into the information history of a subject to ensure that information from the past does not conflict with information a subject may wish to access. Third, the policy may conflict with existing access control policies such as DAC. When a rule conflict occurs, the system must decide whether to handle the conflict as an "exception" and permit access, or to logically "AND" the rules together and deny access.

Chinese Wall is not the first (or last) policy that introduces conflicts between rules, policies, and real world needs. "Exceptions" to security policies are commonplace in order to make a system work in the real world. For example, if Bell-LaPadula were strictly enforced the president would never be able to issue press releases. Therefore, a trusted process is used that violates Bell-LaPadula (an exception) in a secure manner in order to pass non-sensitive messages from the president to the public.

The problem now is identifying valid exceptions, ensuring these exceptions are not used to implicitly leak sensitive information, and ensuring that exceptions do not conflict or amplify a weakness in security. As a system matures, users will compound the problem by demanding additional exceptions to make a system more usable.

To address these problems, Mitre is developing a Generalized Framework for Access Control (GFAC)[23]. GFAC provides a flexible structure that aids in the construction of any type of access control policy. There are no exceptions in GFAC because all interactions between subjects and objects must be explicitly stated. Most existing security policies are consistent with GFAC. The remaining discussion will detail these advantages as well as the structure and components of GFAC.

GFAC is composed of 4 components[23]:

- Access Control Information (ACI) - Characteristics or properties of subjects and objects. Their names are used in specifying the rules of the system; their values are used by the access control rules.
- Access Control Context (ACC) - Additional information, such as time of day, used in access control decision making.
- Access Control Rules(ACR) - The set of formal expressions of policy for adjudicating requests by subjects for access to objects.
- Access Control Authorities (ACA) - Agents who specify ACI, ACC, and ACR.

Given just ACI, ACR and ACA, one can specify policies such as DAC, and Bell-LaPadula. ACI describes ids and security levels, ACR is the set of rules that moderate the interaction of subjects to objects based on ids and/or security levels, and ACA specifies who is permitted to update the security database. The real power of GFAC comes in the form of ACC. This permits the designers of security policies to incorporate any needed external information into the construction of the policy: time of day, operational mode of the system, age of data, history of subject, conditions for downgrade, etc. With this external information, policy designers can specify complex ACRs such as:

- Subject X can access object Y between the hours of 4:00-5:00.
- Subject X can access object Y when Y is 10 years old.
- Subject X can access object Y if X does not contain Z in its information history.
- Subject X can access object Y if the system is in war mode.

These rules exemplify the flexibility of GFAC. With standard security policies, exceptions exist because the policy is so rigid that it cannot cover all possible circumstances. With GFAC, any and all external information can be stated in the policy so there is no reason for exceptions to exist. GFAC's flexibility also permits security policies such as Clark-Wilson, Bell-LaPadula, Type Enforcement, and DAC to be stated in terms of GFAC. In addition, it is possible for all these policies to co-exist because GFAC can contain rules for combining rules (meta-rules). Examples of meta-rules include inheritance, logical AND, and explicit control over resolving rule conflicts.

Because of this flexibility, GFAC is ideal for describing adaptive security policies. Adaptive policies are dynamic in nature because they moderate a system that is dynamic. GFAC permits users to include the dynamic attributes of the system into the policy and then develop a set of rules based on those attributes. The C^2 example from section 2 could be described in terms of GFAC:

- ACI: Roles such as OP.COA.HQ, objects such as Transportation Requirements, levels such as Top Secret.
- ACC: System modes such as WAR, PEACE, TRAINING.
- ACR:

```
if the system is in WAR mode then
   OP.COA.HQ can read INTELLIGENCE data.
```

Although GFAC provides a framework for addressing both existing security policies and exceptions to these policies, we will not make further use of it in this report because access control policies have a significant flaw with regard to preventing disclosure of sensitive information. We will discuss this flaw and how information flow policies address the flaw in the next section. Then, in Section 5 we will discuss an approach for extending information flow policies to deal with adaptive systems in much the same manner as GFAC allows access control policies to be extended to deal with adaptive systems.

4.2 INFORMATION FLOW POLICIES

In Section 4.1 we thought of a computing system as consisting of active entities (subjects) and passive entities (objects). The motivation behind access control policies is that the data to be protected is stored in objects and the only means available for accessing the data are subjects. The contention is that the information can be protected by restricting subjects to observing only objects they are permitted to observe.

Although the contention is plausible, there are some difficulties in attempting to protect data using access control policies. The main difficulty is that the meaning of the policy is strongly dependent on the meaning of "object" and the meaning of "observing."

Intuitively, objects correspond to files. So, for example, the hardware data registers in a computing system are not normally considered objects. If the data registers are not cleared when a high-level subject is swapped out and a low-level subject is swapped in, then any data loaded into the registers while the high-level subject was executing will be available to the low-level subject. Although this allows a low-level subject to obtain high-level data, it is not normally detected by an access control policy since the registers are not normally considered objects.

Although this problem can be addressed by considering the registers as an object with a security level equal to that of the currently active subject, the point is that access control policies only protect data stored in objects. If some entity that can contain data is not considered an object, then it might be possible to circumvent the access control policy by moving data through that entity.

Even if the problem of defining the set of objects is handled, there is still the question of what it means to observe an object. For example, it seems reasonable to allow a low-level subject to write to a high-level object. If the high-level object has been deleted, then what action should be taken? If any abnormal action is taken, the low-level subject might be able to infer that the high-level object has been deleted. Does inferring that the object has been deleted constitute observing the object?

Both of these problems can be addressed by considering the manner in which data flows in the system being analyzed. Any time that data can flow from an object to an entity, the entity must also be an object. With regard to the hardware registers example, if a subject can load data from an object into the registers, then the registers must be considered an object. So, information flow analysis can be used to identify the set of objects in the system. Any time that data flows from an object to a subject, the subject has observed the object. With regard to the object deletion example, if the status that the subject receives when it writes to a deleted object is different than the status it received before the object was deleted, then the fact that the object has been deleted has flowed to the subject.

Rather than using information flow analysis to determine the set of objects and the notion of observing for an access control policy, it is common when performing information flow analysis to totally replace the access control policy with an information flow policy. All information flow policies are of the following spirit:

> *Actions taken by high-level subjects cannot alter the response that low-level subjects receive from the system.*

In the remainder of this section we consider various formalizations of this concept for deterministic, nondeterministic, and stochastic system models.

4.2.1 Deterministic Systems

The first information flow policies were developed in the context of deterministic systems. A system is deterministic if the actions it will take in the future are totally determined by those it has taken in the past. We can view such a system as a state machine with:

- a set of states, S
- a set of operations, $\mathcal{OP}$
- a set of outputs, $\mathcal{OUT}$
- a rule indicating whether a given operation can be executed in a given state (we will use $valid(op, st)$ to represent that op can be executed in st)
- whenever an operation op can be executed in a state st, a rule indicating:
  - the state resulting from executing the operation (which we will denote by st^{op})

- the output that results from the operation (which we will denote by $output(op, st)$)

In order to state an MLS security policy for the system, it is necessary to assign security levels to the operations and the outputs. We will use $level(op)$ and $level(out)$ to denote the level of an operation and output, respectively.

Intuitively, a system is MLS secure if the actions taken by high-level subjects do not affect the behavior of the system that is visible at the low-level. By equating the actions taken by high-level subjects with operations executed by high-level subjects and the behavior of the system that is visible at the low-level with low-level outputs, this requires that high-level operations not influence low-level outputs.

By extending our notation for single elements to sequences in the natural way (for example, if seq is a sequence of operations, then st^{seq} is the state obtained by applying each of the operations in seq to the state st in order), we obtain the following security policy:

A deterministic system is secure if and only if:

for each l, st, op, and sequence of operations seq such that $valid(seq, st)$:

$$valid(seq|l, st)$$

and $$output(op, st^{seq})|l = output(op, st^{seq|l})|l$$

where $x|y$ is the sequence obtained by removing all of the elements from x having a level that is not dominated by that of y

Such a system will be said to be *noninterfering*. In other words, a system is noninterfering if the low-level outputs caused by an arbitrary sequence of operations are the same as the low-level outputs caused when only the low-level operations in the sequence are executed. This concept of MLS security is essentially the same as that proposed in [27], [31], and [30].

4.2.2 Nondeterministic Systems

It is generally accepted that noninterference is the correct formalization of an MLS information flow policy for a deterministic system. Unfortunately, many of the systems of interest (in particular, distributed C^2 systems such as those being considered on this project) do not appear to be deterministic. For example, it is quite common to view a read operation from shared memory that is simultaneously being written as being nondeterministic. Due to the race condition existing between the read and write operations, the value read could be the value that existed before the write operation was initiated, the value existing after the write operation was completed, or even a merger of the two values depending on which operation completes first. It is not clear that such systems are as nondeterministic as is commonly believed, but for now we accept the common view on this matter. In sections **4.2.2.2** and **4.2.3** we consider the validity of this viewpoint.

Given the assumption that distributed C^2 systems are nondeterministic, it is clear that noninterference for deterministic systems is not the correct formalization of an MLS information flow policy for a distributed C^2 system. Before discussing what we believe to be the correct formalization of an MLS information flow policy for a distributed C^2 system. we will consider policies that have been proposed to date.

4.2.2.1 Previous Work — Earlier attempts at defining MLS information flow policies for nondeterministic systems include Forward Correctability [17], Hook-Up Security [20], Restrictiveness [21], and Strong Noninterference [28]. The common thread through all of these policies is that a system is said to be secure if for any two sequences of actions that can be taken by high-level subjects, the low-level behavior of the system is the same regardless of which sequence of actions is taken. It is our belief that the reason there is such a diversity of policies attempting to say the same thing is that they are all based on the CSP[15] formalism or, in the case of [28], a formalism with semantics similar to CSP, and we believe this formalism hampers the statement of this type of policy. Because of the difficulty in stating such a policy in CSP, none of the policies are intellectually satisfying and it is natural to strive for an alternate statement of security. We will address this issue further in Section **4.2.2.3**.

Rather than discussing all of the aforementioned policies, we will discuss Restrictiveness as a representative example. The main difference between it and the other policies is that it is written in a formalism that, while based on CSP, is really a state-machine formalism while the others are actually written in the CSP formalism. This difference is not as significant as it may seem because (as will be discussed in the next contract deliverable) there is a straightforward connection between the state-machine formalism and the CSP formalism.

As in the deterministic state machine model, $\mathcal{S}$, $\mathcal{OP}$, and $\mathcal{OUT}$ represent the states, operations, and outputs of the machine. The differences are that:

- It is traditional to refer to $\mathcal{OP}$ as the set of inputs.
- Instead of there being a single state resulting from executing *op* in *st*, there is a set of states that can be reached from *st* by executing *op*. We will use $possible(st, st_{new}, op)$ to denote that st_{new} can be reached from *st* by executing *op*.

  Note: $valid(op, st)$ is equivalent to there exists some st_{new} such that $possible(st, st_{new}, op)$.
- Outputs do not have to occur directly after the input generating the output. Moreover, outputs do not even have to be generated by inputs. Instead, outputs are operations generated at the discretion of the system that result in a value being transmitted from the system in addition to possibly causing the state of the system to change. We will use $possible(st, st_{new}, out)$ to denote that the system may generate *out* from *st* and then transition to st_{new}.
- The set of elements in $\mathcal{OP}$ executed at levels dominated by l is denoted by $\mathcal{LI}_l$ and the set of the remaining elements of $\mathcal{OP}$ is denoted by $\mathcal{HI}_l$. When l is clear from the context, we will simply refer to these sets as $\mathcal{LI}$ and $\mathcal{HI}$.

- The set of elements in $\mathcal{OUT}$ occurring at levels dominated by l is denoted by $\mathcal{LO}_l$ and the set of the remaining elements of $\mathcal{OUT}$ is denoted by $\mathcal{HO}_l$. When l is clear from the context, we will simply refer to these sets as $\mathcal{LO}$ and $\mathcal{HO}$.
- The set of system events is the union of $\mathcal{OP}$ and $\mathcal{OUT}$.
- $\mathcal{H} = \mathcal{HI} \cup \mathcal{HO}$ and $\mathcal{L} = \mathcal{LI} \cup \mathcal{LO}$ are the set of high-level events and low-level events, respectively.

Then, a system is said to be Restrictive if:

a. It is input total. This means that $valid(op, st)$ holds for all $st \in \mathcal{S}$ and $op \in \mathcal{OP}$.

b. For each level l there is an equivalence relation ($\approx_l$) on states such that:

(1) If $op \in \mathcal{HI}$ and $possible(st, st_{new}, op)$, then $st \approx_l st_{new}$.

(2) If $op \in \mathcal{LI}$, $st_1 \approx_l st_2$, and $possible(st_1, st_{n1}, op)$, then there exists some st_{n2} such that $possible(st_2, st_{n2}, op)$ and $st_{n1} \approx_l st_{n2}$.

(3) If γ is a sequence of outputs each of which is in $\mathcal{HO}$, $st_1 \approx_l st_2$, and $possible(st_1, st_{n1}, \gamma)$, then there exists some st_{n2} and ω such that:

- ω is a sequence (possibly empty) of outputs from $\mathcal{HO}$
- $possible(st_2, st_{n2}, \omega)$
- $st_{n1} \approx_l st_{n2}$

(4) If α and β are sequences of outputs from $\mathcal{HO}$, out is in $\mathcal{LO}$, $st_1 \approx_l st_2$, and $possible(st_1, st_{n1}, \alpha + < out > + \beta)$, then there exists some st_{n2}, δ, and ω such that:

- δ and ω are both sequences (possibly empty) of outputs from $\mathcal{HO}$
- $possible(st_2, st_{n2}, \delta + < out > + \omega)$
- $st_{n1} \approx_l st_{n2}$.

Here, $st_1 \approx_l st_2$ is meant to signify that there are no differences between st_1 and st_2 visible below l. So, the intuition behind b is:

a. Operations executed by high-level subjects should not change low-level information.

b. Operations executed by low-level subjects should only change low-level information in ways determined by low-level information.

c. High-level outputs are only visible to low-level subjects through changes to low-level information and the changes are determined by low-level information.

d. Low-level outputs are determined by low-level information and system actions that are independent of high-level inputs.

The main disadvantages of Restrictiveness are that:

- Rather than making an assertion about arbitrary sequences of operations, it attempts to make assertions about operations in isolation. Although this makes it much easier to show that a system is Restrictive, it is unsatisfying in that it is not clear what assertion about arbitrary sequences of operations follows from the system being Restrictive. It is more desirable to start with a global statement of security and derive conditions on each of the operations (such as those required by Restrictiveness), there is really nothing wrong with starting with conditions on each of the operations, deriving a global security policy, and providing the justification for claiming that the global policy is the "correct" definition of security. For example, we conjecture that Restrictiveness is equivalent to requiring that a system P be input-total and satisfy the following requirement (stated in the CSP formalism):

  Given any traces τ_1 and τ_2 such that τ_1 and τ_2 differ only in events in $\mathcal{H}$:

  $$\{ x|(L \cup \mathcal{HI}) : x \in traces(P/\tau_1)\} = \{x|(L \cup \mathcal{HI}) : x \in traces(P/\tau_2)\}$$

  where:

  * The set $traces(Q)$ is the set of possible execution histories (sequence of operations and outputs) of the system Q.
  * P/s denotes the system that behaves like P behaves after s occurs in P. P/s can be thought of as defining the future behavior of the system after s occurs.

  In other words, for any two sequences of activities that could occur in the system that appear the same at the low-level, the future behavior visible at the low-level is the same.

- The conditions dealing with outputs are not very intuitive. If the low-level information in st_1 is the same as that in st_2, it seems reasonable to require that any low-level output that can occur in one state can occur in the other state. Instead, Restrictiveness allows for the states to have different outputs as long as the system can take appropriate action (through high-level outputs) to correct the discrepancy. For example:

  Suppose:

  * The only data in the system is an integer N.
  * There are high-level inputs h_1-h_{100} that can be used to set N to any value between 1 and 100.
  * There is a high-level output that increments the value of N as a side-effect.
  * The only other high-level output decrements the value of N as a side-effect.

* There are low-level outputs o_1-o_{100} corresponding to the value of N.

It is straight-forward to show that this system is Restrictive. The general idea is that the only possible illicit information flow would be through a high-level subject setting N to some value and a low-level subject observing the setting of N through a low-level output. But, the system can always undo the setting of N by taking appropriate action. For example, if the high-level input results in N being changed from 1 to 100, the system can correct by generating 99 decrementing outputs.

On the other hand, if the system chooses to do almost anything else (for example, not generating any outputs), then the change will be noticeable.

This problem can be addressed by following our intuition and requiring that whenever $st_1 \approx_l st_2$:

* A low-level output can occur in st_1 if and only if it can occur in st_2.
* If st_{n1} and st_{n2} are the states resulting from a given low-level output occurring in st_1 and st_2, respectively, then $st_{n1} \approx_l st_{n2}$.

Then:

* Since o_n can occur exactly when $N = n$, our new definition of security requires that the value of N is the same in both states whenever $st_1 \approx_l st_2$.
* Regardless of the value of N in a given st, there is some input h_n that will result in the value of N being altered. Thus, Restrictiveness' requirement that high-level inputs not cause changes visible at the low-level is not satisfied.

So, our intuition that the system is not secure is supported by this definition of security. We will refer to a system that is Restrictive and satisfies this additional requirement as being *Strongly Restrictive*. Strong Restrictiveness does not totally address the problem, though. For example, consider the following variant of the above example system:

> Suppose that instead of there being increment and decrement outputs, there are high-level outputs $n_1 - n_{100}$ that can set N's value just as the high-level inputs can.
>
> Suppose that any h_k received after a n_j and before the next o_m is ignored.

The system can more formally be described as:

> Suppose each state has variables N, an integer, and R, a Boolean.
>
> If R is set to TRUE, then each h_k is treated as a no-op. Otherwise, h_k sets N to k and does not affect R.
>
> An output n_j can occur at any time. As a result of n_j occurring, R is set to TRUE and N is set to j.

An output o_m can only occur when R is set to TRUE and $N = m$. As a result of o_m occurring, R is set to FALSE.

Then, the following analysis shows that the system is Strongly Restrictive:

* Define $st_1 \approx_l st_2$ if and only if:
  - $st_1.R = st_2.R$, and
  - Whenever $st_1.R$ and $st_2.R$ are both set to TRUE, $st_1.N = st_2.N$
* High-level inputs can only change N and can only do so when R is set to FALSE. Since N is only relevant to the definition of $\approx_l$ when R is set to TRUE, high-level inputs do not change the low-level view of the system.
* Suppose that $st_1 \approx_l st_2$.
  - If o_m can occur in st_1, then $st_1.R$ = TRUE and $st_1.N = m$. Then, $st_2.R$ = TRUE and $st_2.N = m$. So, o_m can occur in st_1 if and only if it can occur in st_2. As a result of o_m occurring R is set to FALSE. So, $st_{n1} \approx_l st_{n2}$, where st_{n1} and st_{n2} are the states obtained as a result of o_m occurring in st_1 and st_2, respectively.
  - Suppose n_j can occur in st_1 and result in st_{n1}. Since n_j can occur in any state, it can occur in st_2. As a result of n_j occurring in a state, R is set to TRUE and N is set to j. So, n_j can occur in st_2 and result in a state st_{n2} such that $st_{n1} \approx_l st_{n2}$.

The same comments as we made about the previous example system can be made about this example system. The security of the system is dependent on the outputs that the system chooses to make. If the system almost always chooses to output n_j such that $j = N$, then it introduces little noise into the system and the low-level outputs correspond closely with the high-level inputs. On the other hand, if the system chooses to make totally random high-level outputs, there is little correlation between the high-level inputs and the low-level outputs.

Our point is that it is quite possible that a nondeterministic model of a system is completely secure while the actual system is almost totally insecure. This is a common problem with existing security policies for nondeterministic systems that will be considered further in sections **4.2.2.2** and **4.2.3**.

4.2.2.2 Strict vs. Loose Nondeterminism— As alluded to earlier, there is a rather serious deficiency in existing nondeterministic security policies. Basically the problem is that it is not always clear how the nondeterminism in the model relates to the nondeterminism in the actual system.

In [11] two types of nondeterminism are described. *Loose nondeterminism* is the type of nondeterminism typically associated with CSP processes. Any implementation that has less nondeterminism than a model is a valid implementation of the model. Since the information flow policies we have described for nondeterministic systems rely on high-level outputs being able to mask high-level

operations, it is quite possible that removing nondeterminism from the model will result in the implemented system becoming insecure. This is a serious concern because the goal is to ensure that the implemented system is secure by showing that the model is secure. If it is possible for the model to be secure and a "valid" implementation of the model to be insecure, than there is little point in analyzing the model.

Strict nondeterminism requires that the model and the implementation have the same degree of nondeterminism. Thus, when performing a security analysis, it is necessary to assume strict nondeterminism. Unfortunately it is not always possible to show that the nondeterminism present in the model is strict. For example, race conditions are often specified as being nondeterministic because it is not clear exactly what will occur as the result of a race condition. Since it is not clear exactly what will occur, it might not be possible to show that each of the possibilities specified in the model can really occur in the implementation.

So, we see that the question of information flow security in nondeterministic systems is significantly more difficult than that for deterministic systems. In Section **4.2.3**, we will see that the situation is even more serious.

4.2.2.3 Difficulties in Using CSP — In Section **4.2.2.1** we stated that we believe there is an inherent problem in using CSP to state information flow policies. Although we do not have a proof that it is not possible to state information flow in terms of CSP, we feel it is worthwhile to briefly mention the problems that we see with using CSP. It is important to note that similar problems exist in many other formalisms. Our main reason for critiquing CSP in particular is that most of the previous security policies for nondeterministic systems use the CSP formalism.

We begin the discussion with a quote from page 24 of [15]:

> In choosing an alphabet, there is no need to make a distinction between events which are initiated by the object ...and those which are initiated by some agent outside the object ...The avoidance of the concept of causality leads to considerable simplification in the theory and its application.

We believe causality to be a necessary concern in an information flow policy. Consider the following example:

> Suppose that the events that can occur in a system are h_0, h_1, l_0, and l_1 with the former two representing high-level events and the latter two representing low-level events.
>
> Suppose that the system is such that l_i can only occur if it is directly preceded by h_i for each i.
>
> If we view h_0 and h_1 as inputs and l_0 and l_1 as outputs, the system is obviously insecure. Depending on whether l_0 or l_1 is output, it is obvious which event was last input by the high-level subject.

If we view h_0 and h_1 as outputs, then the system is obviously secure (since there are no inputs, the only information that is output from the system is the outputs that it nondeterministically chooses to make). Note that we assume here (as we have earlier in this report) that the goal is to protect high-level inputs rather than high-level outputs. We will address the protection of high-level outputs in Section **4.2.5**.

The only distinction between random noise and actions issued by high-level processes is given by the division of the high-level events into inputs and outputs. Yet, [15] explicitly states that the semantics of CSP are not adequate to distinguish between inputs and outputs.

The reason this complicates the statement of an MLS information flow policy is that it is difficult to distinguish between high-level outputs that are caused by low-level inputs, those caused by high-level inputs, and those that are randomly generated by the system. We illustrate this with the following examples:

- The most straightforward generalization of noninterference is to simply require that given any trace τ of the system, if τ_l is τ with all high-level events removed, then τ_l is a trace of the system. The following example shows that this would exclude some obviously secure systems:

  Suppose that a system has a low-level input i_l, a high-level output o_h, and a low-level output o_l.

  Suppose that any trace of the system is such that every o_l is preceded by an o_h and every o_h is preceded by an i_l.

  $\tau =< i_l, o_h, o_l >$ is a trace of the system while $\tau_p =< i_l, o_l >$ is not. Yet, the system is obviously secure since it does not have any high-level inputs.

- Since it is not reasonable to require that the system work the same with all high-level events removed, we must determine which high-level events are reasonable to remove. Clearly it should not be necessary for high-level input events to be present in order for a system to be secure, so the next natural step is to attempt to define security as in the previous step with τ_p meaning τ with high-level inputs removed. But, this is too restrictive, too.

  Suppose that a system has a high-level input i_h, and a high-level output o_h.

  Suppose that any trace of the system is such that every o_h is preceded by an i_h.

  $\tau =< i_h, o_h >$ is a trace of the system while $\tau_p =< o_h >$ is not. Yet, the system is obviously secure since it does not have any low-level events.

- Since the first attempt failed because too many outputs were removed in going to τ_p (o_h should not have been removed) and the second fails because not enough were removed in going to τ_p (o_h should have been removed), the next natural step is to only remove some high-level outputs. One possibility is to require that given any τ_p that differs from τ only in high-level inputs, there exists a τ_c that differs from τ_p only in high-level outputs such that τ_c is a trace (this policy is known as Correctability[17]). The intuition behind this policy is that no matter how high-level subjects "perturb" the system through high-level inputs, the

system can always "correct" itself through high-level outputs so that the high-level inputs are not visible to low-level subjects. The difficulty with this policy is that it requires the system to predict the future in order to maintain security. Since it is difficult to completely list the traces for a good example, we will simply give a sketch of the system.

Suppose that a system allows a high-level subject to input a 0 or a 1, and outputs a 0 or a 1 at the low-level. If no value is input, then the low-level output is 0. Otherwise, the low-level output is equal to the high-level input if the system chooses not to introduce noise and the opposite of the high-level input, otherwise. Suppose that we require the system to decide whether to introduce noise before the high-level input is made. Regardless of the high-level input made, the system can cause the low-level output to be 0 by appropriately deciding whether to introduce noise. But, the system cannot determine whether it should introduce noise without knowing the high-level input that will be made in the future.

For example, if the high-level subject observes the system introducing noise, then it inputs the opposite of the value it wishes to send. Otherwise, it inputs the value it wishes to send. In either case, the low-level output is the value that the high-level subject wishes to send.

Another way of making the same point is that the proposed definition of security assumes that the system knows all of the inputs that will be made by the high-level subject and takes appropriate action to mask these inputs, while in reality it is the high-level subject that knows the actions taken by the system and can mask them through appropriate outputs.

- So, it is clear that it is not reasonable to allow arbitrary "corrections." The next natural step is to only allow "corrections" based on past occurrences. This is essentially what Restrictiveness and Forward Correctability require. As noted at the end of Section **4.2.2.1**, this has the drawback of calling some systems secure even though they are insecure if the system does not introduce noise at the correct time. Since we have already made use of the input/output distinction and the high/low distinction, it is not clear how to change the policy so that it does not allow for such systems to be secure. We suspect that such a change is not possible in CSP because the semantics do not allow for the concept of causality.

Further research needs to be done to determine whether there really is an inherent difficulty in defining information flow policies in CSP. If there really is, then it is necessary to determine whether there is actually an inherent difficulty in defining information flow policies for nondeterministic systems regardless of the formalism selected. In the next section we will consider an extension to nondeterministic systems that results in a much more complicated system model but seems to address some of the problems encountered in the system model for nondeterministic systems.

4.2.2.4 Summary — In this section we have discussed previous attempts at defining information flow policies for nondeterministic systems and their deficiencies. We have proposed a new security policy, Strong Restrictiveness, that we believe addresses some of these deficiencies. We

have also discussed our opinion that certain formalisms seem to hamper the statement of an information flow policy. One issue that has not been addressed in this section is the likelihood of events occurring. In nondeterministic models, no distinction is made between events that are quite likely to occur and events that are quite unlikely to occur. In the next section, we will discuss the implications this has for a security analysis.

4.2.3 Stochastic Systems

Recently some consideration has been given to information flow policies in stochastic systems[16]. The only difference between a stochastic system and the nondeterministic systems introduced in the previous section is that instead of simply recognizing that an action can result in more than one output, we also consider the probability of each result occurring. We can then define the *stochastic information flow policy* to require that:

> For any sequence of inputs and low-level outputs, the probability that the system will generate the given sequence of low-level outputs from the sequence of inputs is the same as the probability the system will generate the low-level output sequence from only the low-level inputs of the given input sequence.

For example, we can add probability distributions to one of our examples from Section **4.2.2.1** to obtain the following system:

- Suppose :
  - The only data in the system is an integer N.
  - There are high-level inputs h_1-h_{100} that can be used to set N to any value between 1 and 100.
  - There is a high-level output that increments the value of N as a side-effect.
  - The only other high-level output decrements the value of N as a side-effect.
  - There are low-level outputs o_1-o_{100} corresponding to the value of N.
  - Initially, the system attempts to accept a high-level input with probability p and outputs the value of N with probability $1-p$. If the system attempts to get input and no input has been made, then the system does not change state.
  - If the system attempts to accept input and an input has been made, it transitions to a new state from which it can increment N with probability q, decrement N with probability r, accept another input with probability s, or output N with probability $1-q-r-s$. If the system chooses to output N, then the system returns to the initial state. Otherwise, the system repeatedly chooses to increment, decrement, accept new input, or output until an output is made.

The system can be thought of as a Markov chain. In particular, it is basically a drunken-sailor's walk (thinking of incrementing as stepping left, decrementing as stepping right, and outputting as falling down, the system can be thought of as modeling the location at which the sailor will fall down). In order for the system to be secure, the probability of the sailor falling at a given location must be independent of the start location (since the high-level input can be thought of as defining the starting location for the sailor).

It is interesting to note that although this system is Restrictive, the high-level input can have a dramatic effect on the probability with which each output occurs. For example, if h_1 is the only input, then at least 99 increments are required for o_{100} to occur. Since the probability of an increment occurring at any given time after the input is q, the probability of o_{100} occurring after h_1 is input is less than q^{99}. On the other hand, if h_{100} is the only input, the system can immediately output o_{100} with probability $1 - q - r - s$. Consequently, the probability of o_{100} occurring after h_{100} is at least $1 - q - r - s$. If we choose $q = r = s = .1$, then we see that the probability of o_{100} occurring after h_1 is input is essentially 0 while the probability of it occurring after h_{100} is input is at least .7.

In order for the system to be completely secure, the probability of o_{100} occurring should be the same regardless of the high-level input. So, letting x represent the probability of o_{100} occurring, we see that:

- $q^{99} > x > 1 - q - r - s$

Unless q is very close to 1, q^{99} is very close to 0 and $q + r + s$ must be close to 1 for the system to be secure. If q is very close to 1, then $q + r + s$ must be close to 1 since it must be greater than q and less than 1. Consequently, $q + r + s$ must be very close to 1 in order for the system to be secure. In fact, some simple Markov analysis shows that it is impossible to make the system completely secure unless $q + r + s$ is exactly 1. This really is not a valid choice for $q + r + s$ because it means that the low-level output can never occur while our nondeterministic model of the system assumes that it can occur.

In other words, even though the system is Restrictive, it does not satisfy the stochastic information flow policy (regardless of how the probabilities are assigned). This agrees with our intuition in the preceding section that Restrictiveness calls some "insecure" systems "secure."

More research needs to be done to determine whether there are any systems that satisfy Strong Restrictiveness and yet fail to satisfy the stochastic information flow policy regardless of how the probabilities are selected. If there are not any such systems, then it would be possible to use a nondeterministic model with Strong Restrictiveness as the security policy as a preliminary investigation of the security of a nondeterministic system. Then, further analysis of the system could concentrate on refining the model to a stochastic model and determining values for the probabilities that will result in the system satisfying the stochastic information flow policy. If there are some systems that satisfy Strong Restrictiveness but do not satisfy the stochastic information flow policy, then there is the danger that the preliminary analysis would not detect any insecurities while there is no possible way to assign probabilities to make the system secure in the subsequent analysis.

The stochastic information flow policy is basically that given in [16]. There are some serious difficulties with using this policy.

a. It is not clear how the probabilities of events occurring are determined. In [16], it is assumed that system designers will specify the probabilities and leave it to the system implementers to ensure that the implementation is consistent with the design. This would seem to place a great burden on the implementers since some of the probabilities will undoubtedly be influenced by factors beyond the control of the implementers. Another approach would be to assume that the analysis is done after the implementation is complete in which case experimentation could be used to obtain an approximation of the probabilities.

b. Assuming that all of the probabilities can be obtained, there is the issue of performing the analysis itself. If there is much more nondeterminism than determinism present in the system, the analysis will probably become infeasible.

c. Even if experimentation suggests a given probability distribution is appropriate, there is still the same type of concern as discussed in Section **4.2.2.2**. In particular, even though the system appears to be stochastic, it is possible that the system is actually deterministic with transition rules that are so complex that they appear stochastic. This has serious implications when we begin to consider the composability issue in Section **6.1.1**.

4.2.4 Self Evolving Systems

All the information flow policies discussed previously suffer from a fundamental flaw:

> They assume that the correct model of the system is that subjects exist external to the system, send input to the system, and obtain output from the system.

If we replace "subject" with "user", then the above view is reasonable. The point is that while users exist external to the system, subjects exist internal to the system. Since system operations can modify the state of the system, it is possible for system operations to modify the behavior of subjects. On the other hand, system operations do not change the behavior of a user. Consider the following system:

> Suppose that a low-level subject can input a 0 or a 1 and receives an output echoing its input.
>
> Suppose that a high-level subject can input a 0 or a 1, and as a result of its input, it causes the low-level subject to input the same value. If the high-level subject has not yet made an input, assume that all the low-level subject can input is a 0. A case in which this could occur in the real world is that in which the high-level subject's input alters the code object for the low-level subject. In our particular example, the low-level subject is instructed by its code object to input a 0 and the high-level subject can change the low-level subject's code object through an input. System models usually abstract away the fact that a subject executes the sequence of instructions contained in its code object rather than executing any instruction sequence it desires. Thus, unless there are specific operations that allow low-level subject's

to observe the executable object, the executable object appears as an unreadable object in the model and there is consequently no security violation detected when a high-level subject modifies it.

Since the models we have considered previously assume that there is no correlation between high-level inputs and low-level inputs, the system appears to be secure even though the correlation between high-level inputs and low-level inputs clearly makes the system insecure. This example suggests that a way to address this problem would be to attempt to identify instances in which high-level inputs can affect the low-level inputs that may occur next.

Another possible way to address this problem is to view the system as self-contained and compare the manner in which it evolves with the manner in which it would evolve if all of its high level processes and data were purged. In the example above, if the high-level subject's code object instructs it to input a 1, then the result of running the machine would be that the high-level subject would input a 1, the low-level subject would input a 1, and the system would output a 1 to the low-level subject. On the other hand, if we purge the high-level data and subjects, then the low-level subject's executable object would not be altered and the result of running the machine would be that the low-level subject would input a 0 and the system would output a 0 to the low-level subject. Since the operation of the system that is visible at the low-level is different when the high-level data and subjects are purged, the system is seen to be insecure.

In [32], the approach taken is to attempt to determine conditions to place on the processing of instructions that when combined with an information flow policy address the possibility of high-level inputs having an effect on low-level inputs. At the Rome Laboratory 1990 Technology Exchange Meeting, Tanya Korelski mentioned that her group had developed a security policy called "state partitioned state machines" that sounds similar to our approach in which the system is considered to evolve on its own rather than having external inputs. Future research is needed to determine the relationship between "state partitioned state machines", the approach described in [32], and our approach.

4.2.5 Nonobservability

Another flaw present in information flow policies is that they only address the flow of information from high-level subjects to low-level subjects and consequently ignore information flow from high-level objects to low-level subjects. In other words, they do not necessarily prohibit low-level processes from directly reading high-level objects.

This is only a problem when the high-level object is nonmodifiable. In that case, even though a low-level subject might be able to read the object, high-level subjects cannot transmit information through the object due to their inability to modify the object. In some ways this is a less serious security threat than those addressed by information flow policies because the information compromised is limited to that initially stored in the object. On the other hand, the rate at which the information is compromised could be significantly higher for this type of security flaw. For example, if the low-level subject can directly read the high-level object, then it can probably obtain

the information more quickly than it would if the high-level subject had encoded the information in a form suitable for transmission through a covert channel. The encoding and decoding consume time and covert channels themselves typically have a lower bandwidth than the read and write operations in the system.

Regardless of how serious these threats actually are, it makes little sense to ignore them in the security analysis because we can address them with an information flow policy by making a slight modification to the policy. Before describing the necessary modification, we will describe a separate policy, which we call a *nonobservability policy*, to address this issue.

> A system satisfies a nonobservability policy if, regardless of the values assigned to high-level data in the initial state of the system, the outputs resulting from the execution of low-level inputs are the same.

Any system that allows a low-level subject to read a high-level object will not satisfy the nonobservability policy because altering the value of the high-level object in the initial state will result in a different value being read when the system is run.

Now, the obvious way to combine this with the information flow policy is to require that:

- Regardless of the values assigned to high-level data in the initial state of the system, the outputs resulting from the execution of low-level inputs are independent of high-level inputs.

In the case of deterministic systems, this can be formalized as the requirement that:

$$st_1 \approx_l st_2$$
$$\text{and } valid(seq|l, st_1)$$

$$\Rightarrow$$

$$valid(seq|l, st_2)$$
$$\text{and } output(op, {st_1}^{seq})|l = output(op, {st_2}^{seq|l})|l$$

where $st_1 \approx_l st_2$ is used to denote that the only difference between st_1 and st_2 is in data that is not visible at or below l. In [10] this is referred to as *strong noninterference.* This is an unfortunate choice of names since there are at least two other security policies referred to as "strong noninterference" that have no relation to this policy. One developed by ORA is described in [21], and the other was developed by SRI and is described in [28].

Before closing this matter, we would like to discuss the additional complexity added by considering observability. One obvious difference between strong noninterference and noninterference is that it is necessary to define the notion of two states looking the same at a given level. Although this is generally a difficult task, we maintain that it is no more difficult to perform an analysis with respect to noninterference than it is with respect to strong noninterference since we believe that the most reasonable way to perform a noninterference analysis is to divide it into four steps:

a. Determine the appropriate definition of $st_1 \approx_l st_2$.

b. Show that whenever $st_1 \approx_l st_2$, ${st_1}^{seq} \approx_l {st_2}^{seq|l}$, for all seq, and l.

c. Show that the only outputs that can be caused by an input are outputs at levels greater than or equal to that of the input.

d. Show that whenever $st_1 \approx_l st_2$ and op is a low-level operation,

(1) $valid(seq, st_1) \Rightarrow valid(seq|l, st_2)$

(2) $output(op, st_1)|l = output(op, st_2)|l$

The intuition is that if it can be shown that:

- high-level inputs neither affect the view that low-level subjects have of the system nor cause low-level outputs
- the low-level outputs caused by a low-level input are totally determined by the view that low-level subjects have of the system,

then high-level inputs cannot have any influence on low-level outputs. As described in [30], the definition of $st_1 \approx_l st_2$ and the noninterference analysis can be performed simultaneously in a relatively efficient manner. It just so happens that this approach for showing that a system satisfies noninterference actually shows that a system satisfies the stronger property of strong noninterference. So, as long as this seems to be the most efficient way of performing a noninterference analysis, it is pointless to use noninterference as the policy instead of strong noninterference. With regard to extending this work to nondeterministic systems, we note that our Strong Restrictiveness addresses nonobservability in the same way that strong noninterference does. In fact, both Restrictiveness and Strong Restrictiveness are natural extensions of Strong Noninterference, as described in [10], to nondeterministic systems.

It is important to note that strong noninterference does not totally solve the problem of low-level subjects being able to observe high-level objects. It is possible to define $st_1 \approx_l st_2$ in such a way that it requires the value of the high-level data to be the same. After doing so, the reading of this object would no longer be detected as a security violation by the strong noninterference policy. The problem here is that $st_1 \approx_l st_2$ is supposed to capture the intuitive notion of two states appearing the same to subjects at or below l, but we have defined it so it requires information above l to be the same. This conflicts with the intuition of $st_1 \approx_l st_2$ only capturing the low-level data. All this means is that as $st_1 \approx_l st_2$ is constructed, care must be taken to ensure that no high-level data is referenced. This is no different from the care that must be taken to assign correct security levels to each state component in a traditional covert channel analysis[22].

4.3 RELATION WITH EXAMPLE SYSTEM

Although the example in Section 2 does not go into much detail concerning the security policies described in this section, these policies are very important in C^2 systems. Access control policies

are quite often used as the primitive security mechanism on each of the nodes. By building on the access control policies provided by each node, it is possible to show that the node satisfies an information flow policy (unless, of course, it allows information to be transmitted downward in security level). Furthermore, by building on the information flow policy enforced by each node, it is possible to obtain an information flow policy for the entire system (unless, of course, it allows information to be transmitted downward in security level). These policies are not discussed much in Section 2 because we wanted to emphasize the adaptive components of the system, not because they are not important in the analysis of C^2 systems. The Role Flow Rule in Section 2 can be thought of as either a Clark-Wilson policy or a type enforcement policy, while the Mode Rule has aspects of both an information flow policy and access control policies. As further support for the usefulness of these policies, we will show in the next section that they provide a natural foundation for adaptive security policies.

4.4 SUMMARY

In this section we have considered the natural progression of security models. The earlier security models were primitive access control policies. Later, information flow policies were developed for deterministic systems. Next, attempts were made to extend these policies to nondeterministic systems. Finally, work is currently being done to obtain a correct definition of information flow security for nondeterministic systems and a start has been made on defining information flow security for stochastic systems. We have attempted to explain the flaws with each of the earlier security policies and the manner in which subsequent policies address the flaws. We have proposed Strong Restrictiveness as being a more correct definition of information flow security for nondeterministic systems and attempted to explain how it addresses flaws in other proposed definitions. In our examination of nondeterministic systems, we discussed the possibility that we might not yet have the correct formalism for stating information flow policies for such systems. We have also proposed a statement of information flow security for stochastic systems that is similar to that proposed by others. In the future, we hope to show that our definition of information flow security for stochastic systems is a natural extension of Strong Restrictiveness, explore differences between our definition for stochastic systems and definitions proposed by others, and try to identify formalisms that are more conducive to the statement of information flow policies for nondeterministic systems.

SECTION 5

ADAPTIVE SECURITY POLICIES

One of the specific goals of the ASCM project is to consider *adaptive security policies.* The following is taken from the Statement of Work:

> One category of service that must be made available in future C2 systems is that of adaptive policies, i.e., maintaining an acceptable level of information security when there is a desire or need to vary the security policy with time and other external conditions. There are several reasons for considering adaptive policies, such as maintenance access, reconfiguration of system resources, reclassification of information, broadcast messages, and changing operational modes.

Our interpretation of the need for adaptive security policies is that existing security policies are too restrictive for use with real systems. In particular, although requiring that there be no information flow from high levels to low levels seems like a reasonable definition of MLS security, it does not recognize that real systems often need to transmit information from high levels to low levels. For example, a broadcast message from a low-level subject will often require acknowledgements from high-level subjects in order to determine whether the message should be rebroadcast. In this sense "adaptive security policy" is actually a misnomer. It is not that the security policy of the system has changed to allow the normally disallowed information flow downward in level. The policy of the system remains constant, but instead of requiring that there be no downward flow of information, it requires that there be no downward flow of information except for certain special cases. With this interpretation, adaptive security policies are seen to be *conditional security policies.*

5.1 CONDITIONAL SECURITY POLICIES

Given our earlier contention that an information flow policy is the correct notion of security, the first step in developing "adaptive" security policies is to develop a conditional information flow policy. [31] provides such a policy for deterministic state machines by generalizing noninterference to *conditional noninterference.* The policy can be stated as:

> High-level subjects can only interfere with low-level subjects through certain sequences of operations.

In other words, while the noninterference policy requires that there be no interference from high-level subjects to low-level subjects through any sequence of instructions, the conditional noninterference policy weakens this to allow interference through some defined set of sequences. Thus, the policy says that unless one of the exceptional sequences is executed, there is no illicit information flow. Once the system is shown to satisfy the conditional noninterference policy, it remains only to

consider the risk associated with each of the exceptional sequences. In this section we will consider some general classes of exceptional sequences. These classes represent functionality that is desirable to have in C^2 systems but appears to conflict with MLS information flow policies.

5.2 CLASSES OF EXCEPTIONS

The classes of exceptional processing that we will consider are:

a. Reclassification of processes and data.

b. Reconfiguration of system resources.

c. Broadcast messages.

d. Change in operational mode.

5.2.1 Reclassification of Processes and Data

Reclassification refers to changing the security level associated with a piece of information. Many enforcement mechanisms ignore the semantics of information and consider only the object containing the information. Similarly, the sensitivity of a program is usually assumed to be dominated by that of any subject executing the program. So, we will concentrate on the changing of the security level of a subject or object throughout this section rather than the more general problem of reclassification of data. At the end of this section, we will briefly consider the more general problem in the context of a specific interpretation of upgrading.

Many security models assume that the security levels of subjects and objects remain static. Consequently, the reclassification of a subject or object will result in the assumptions of the model being violated, regardless of the direction of the reclassification. In general, there is no reason to prevent a low-level subject from reclassifying information to a higher level. Thus, rather than simply viewing such a reclassification as an exception to an information flow policy, we feel it is more reasonable to develop an information flow policy that is applicable to systems that allow subjects and objects to change security level. After doing so, we will consider the cases that are actually security problems. Finally, we will consider an alternative interpretation of upgrading that addresses the reclassification of information rather than the reclassification of subjects and objects.

5.2.1.1 Nontranquil Information Flow Policies— Systems in which subjects and objects have static security levels are said to satisfy a *tranquility property*. In this section we will consider systems that violate tranquility. Examining our earlier definition of information flow policies, we see that the only reference to the level of a subject or object is the assumption that a level is associated with each input and output. In the policies we considered earlier, the level associated with an input or output was assumed to be independent of the state of the system. So, to adapt our earlier definitions of information flow policies to nontranquil systems, we must make the level

of an input or output a function of the state of the system. Although this can be accomplished in a pure CSP model by making the level of an input or output a function of the trace to which the input or output is to be added, it will be easier in this section to adopt a state machine formalism. For example, Strong Restrictiveness can be stated as:

a. The system is input total.

b. For each level l there is an equivalence relation ($\approx_l$) on states such that:

(1) If $possible(st, st_{new}, op)$ and $op \in \mathcal{HI}(st)$, then $st \approx_l st_{new}$.

(2) If $st_1 \approx_l st_2$, $possible(st_1, st_{n1}, op)$, $op \in \mathcal{LI}(st_1)$, and $op \in \mathcal{LI}(st_2)$, then there exists some st_{n2} such that $possible(st_2, st_{n2}, op)$ and $st_{n1} \approx_l st_{n2}$.

(3) If $st_1 \approx_l st_2$, $possible(st_1, st_{n1}, \gamma)$, and γ is in $\mathcal{HOS}(st_1, st_{n1})$, then there exists some st_{n2} and ω such that:

- ω is in $\mathcal{HOS}(st_2, st_{n2})$
- $possible(st_2, st_{n2}, \omega)$
- $st_{n1} \approx_l st_{n2}$

(4) If $st_1 \approx_l st_2$, $possible(st_1, st_{n1}, < out >)$, and out is in $\mathcal{LO}(st_1)$, then there exists some st_{n2} such that:

- $possible(st_2, st_{n2}, < out >)$
- $st_{n1} \approx_l st_{n2}$.

(5) If $st_1 \approx_l st_2$, then $\mathcal{LI}(st_1) \cup \mathcal{LO}(st_1) = \mathcal{LI}(st_2) \cup \mathcal{LO}(st_2)$

Here we have used $\mathcal{HI}(st)$, $\mathcal{HO}(st)$, $\mathcal{LI}(st)$, and $\mathcal{LO}(st)$ to represent the high-level inputs, high-level outputs, low-level inputs, and low-level outputs with respect to st and $\mathcal{HOS}(st_a, st_b)$ to represent the set of output sequences α such that:

- $possible(st_a, st_b, \alpha)$
- Each out in α is in $HO(st_{out})$, where st_{out} is the state in which out occurs.

The definition of $\mathcal{HOS}$ is more complicated since the level of an output in the middle of a sequence can be altered by outputs preceding it.

There are two differences between this policy and Strong Restrictiveness:

a. The sets $\mathcal{HI}$, $\mathcal{HO}$, $\mathcal{LI}$, and $\mathcal{LO}$ are state dependent.

b. We require that the set of low-level events in st_1 be the same as the set of low-level events in st_2 whenever $st_1 \approx_l st_2$. This is a natural requirement since $\approx_l$ is intended to capture the portions of the system that are visible at the low level.

This straightforward generalization of Strong Restrictiveness to nontranquil systems prohibits the downgrading of information while allowing information to be upgraded by low-level subjects. The information flow policy developed for stochastic systems in Section **4.2.3** can be generalized in the same manner.

5.2.1.2 Exceptions — Inputs or outputs that do not satisfy the requirements of our generalization of Strong Restrictiveness may result in information being transmitted downward in security level. Since information flow policies generally prohibit such information flow, such inputs and outputs must be viewed as exceptions to the information flow policy. In the following we will divide the exceptions into two classes, namely *pseudo-downgrades* and *true-downgrades.*

5.2.1.2.1 Pseudo-Downgrades — In some cases, the downgrading of information does not constitute any security threat. In particular, the use of *trusted subjects* for performing downgrades can in some cases eliminate the danger of information being transmitted downward in security level.

The key point to understanding this section is that the level of a piece of information is defined by users rather than the system. The level that the system assigns to a piece of information should be at least as high as the level that the user assigns to the information. This requirement is typically enforced by:

a. Requiring the user to inform the system of any information input to the system.

b. Requiring the system to store the information in a data container with a level at least as high as the level provided by the user.

c. Preventing information from flowing downward in security level.

If these three rules are followed, then the level of a data container always dominates that of the contained information (this is the objective of the Mode Rule in Section **2**). So, the enforcement of an information flow policy is simply the means to an end. As long as the level the system assigns to a piece of information is always at least as high as that the user assigns to the information, there is no threat of data compromise regardless of whether the system satisfies an information flow policy. In particular, a user who is authorized to operate at both the secret level and the unclassified level is permitted to read information at the secret level and type it back in at the unclassified level. This is not viewed as a security violation since the user is *trusted* not to enter information at an incorrect level. So, if the user reenters secret information at the unclassified level, it is assumed that the user has determined that the information does not need to be classified any higher than unclassified. In other words, the user has reclassified the information.

One special case is that in which a trusted process downgrades information that is a function of information that is already available at the low-level. In this case, the trusted process can be viewed as being at the low-level in the security analysis, and it is not even necessary to conditionalize the policy. We can use our generalized form of Strong Restrictiveness with the inputs and outputs associated with trusted processes included in $\mathcal{LI}$ and $\mathcal{LO}$. We will refer to the security policy obtained from this new interpretation of $\mathcal{LI}$ and $\mathcal{LO}$ as being *TH-Guarded*. Assuming that Trojan horses are not present in trusted code, then our policy requires that the system guard against Trojan horses by eliminating the possibility of untrusted subjects causing information to flow downward in security level.

A good example of this is terminal I/O in the LOCK system[1]. In LOCK, each terminal has a driver subject residing at device high, and subjects communicate with the terminal through buffer objects that are at the level of the communicating subjects. Input from the terminal is accomplished by the driver subject writing the input data into the buffer object provided by the subject currently in control of the terminal. Thus, terminal input is accomplished by downgrading information. Since the driver subject will only perform the downgrade if the level of the buffer object is the same as that at which the user input the data, this is really a pseudo-downgrade. In fact, it could be argued that the information was upgraded by the driver subject when it was read from the terminal and that the downgrading actually returned the information to its proper level. When viewed in this light, the downgrading is seen to not be a security violation. So, although I/O in LOCK can result in information being transmitted downward in security level, it is TH-Guarded.

Another good example is that of a sanitizer-downgrader. The idea behind a sanitizer-downgrader is that downgrading should be a two step process. In the first step, the system sanitizes the information by eliminating any changes that result from actions that were not initiated by a trusted subject or a user. After eliminating any such changes, the system downgrades the information. If the sanitizer-downgrader is TH-Guarded, then the proposed sanitization is appropriate. Otherwise, we will be able to identify the operations that untrusted high-level subjects can take that are not addressed by the sanitization by determining the reason that TH-Guardedness fails to hold.

It is desirable to analyze the code for trusted subjects to ensure that they behave as specified. For example, if the trusted subject in a sanitizer-downgrader does not sanitize information as described in its specification, the analysis performed on the model will not be applicable to the implementation. Although this is true of any analysis performed on the model, the trusted subjects are prime candidates for close examination due to their privileged status. Analysis of the code for trusted subjects can also be used to provide support for the assumption that there are no Trojan horse subjects present in trusted code.

In general, the greater the structure in the system, the easier it is to distinguish between true downgrades and pseudo-downgrades. For example, in a type enforcing system (or a system enforcing a Clark-Wilson type policy), a finer degree of control is present than in typical MLS systems. As an example, the following can be done on LOCK[1]:

- The set of trusted domains is identified to be the set of domains in which trusted subjects can execute.

- For each trusted domain, an object type is created to represent the executable code for the domain.
- Each trusted domain is prohibited from executing code from any object with a type other than that associated with the domain.
- All domains are prohibited from modifying objects of a type representing the executable object for a trusted domain.

By configuring the system in this manner, we are assured that the code that a trusted subject executes when the system runs is the code that it was intended to execute. This is important since there is little use in analyzing a piece of code for correct behavior if a different piece of code is executed when the system is run.

It is also possible to define types of objects that are only modifiable from trusted domains. For these objects, the type enforcement mechanism prevents untrusted subjects from modifying the objects and consequently reduces the security risk associated with downgrading the objects. Thus, the system enforces an integrity policy on the identified types of objects. The objects are guaranteed by the reference monitor mechanisms not to have been modified by any but a certain class of subjects. And those subjects are guaranteed not to contain any Trojan horses, so we know that the objects contain no high-level information.

5.2.1.2.2 True-Downgrades — It is quite often the case that the downgraded information is determined by factors other than trusted subjects and information already visible at the low-level. A typical example is that of downgrading an object without performing sanitization. If the object has been manipulated by high-level subjects, then it is quite possible that they have inserted information of which the user performing the downgrade is not aware. In this case, the approach suggested in the previous section cannot be used to demonstrate the security of the system because the actions of the untrusted high-level subjects become observable at the low-level as a result of the object being downgraded. Instead, we conditionalize the policy to exclude the trusted operations from the information flow analysis. For example, we can define a system to be *Conditionally TH-Guarded* if:

a. The system is input total.

b. For each level l there is an equivalence relation ($\approx_l$) on states such that:

   (1) If $possible(st, st_{new}, op)$, $op \in \mathcal{HI}(st)$, and $op \notin \mathcal{PRIV}(st)$, then $st \approx_l st_{new}$.

   (2) If $st_1 \approx_l st_2$, $possible(st_1, st_{n1}, op)$, $op \in \mathcal{LI}(st_1)$, and $op \in \mathcal{LI}(st_2)$, then there exists some st_{n2} such that $possible(st_2, st_{n2}, op)$ and $st_{n1} \approx_l st_{n2}$.

   (3) If $st_1 \approx_l st_2$, $possible(st_1, st_{n1}, \gamma)$, and γ is in $\mathcal{HOS}(st_1, st_{n1})$, then there exists some st_{n2} and ω such that:

   - ω is in $\mathcal{HOS}(st_2, st_{n2})$
   - $possible(st_2, st_{n2}, \omega)$

- $st_{n1} \approx_l st_{n2}$

(4) If $st_1 \approx_l st_2$, $possible(st_1, st_{n1}, < out >)$, and out is in $\mathcal{LO}(st_1)$, then there exists some st_{n2} such that:

- $possible(st_2, st_{n2}, < out >)$
- $st_{n1} \approx_l st_{n2}$.

(5) If $st_1 \approx_l st_2$, then $\mathcal{LI}(st_1) \cup \mathcal{LO}(st_1) = \mathcal{LI}(st_2) \cup \mathcal{LO}(st_2)$

where everything is defined as for TH-Guardedness with the addition that $\mathcal{PRIV}(l, st)$ denotes the set of instructions that are privileged to downgrade information from a level not dominated by l to a level dominated by l.

By showing that all of the operations that are not in $\mathcal{PRIV}$ satisfy TH-Guardedness, we reduce the scope of further analysis to the operations in $\mathcal{PRIV}$.

Furthermore, by determining the reason for the failure of the proof of TH-Guardedness for operations in $\mathcal{PRIV}$, we can determine the ways in which the trusted subjects can be exploited by untrusted subjects to transmit information downward in security level. Then, the analyst must make a determination as to the severity of the security risk associated with the downgrading of information. Since the exploitation scenarios and amount of risk are system specific, it is not possible to address them in this document. Our point here is that although the conditional information flow analysis is not the entire security analysis, it can be used to:

- Completely address large portions of the system.
- Identify portions of the system that need further analysis.
- Aid in the analysis of these portions of the system by identifying the ways in which untrusted subjects can exploit trusted subjects.

5.2.1.2.3 Complete Upgrade — Before concluding our discussion of reclassification, we discuss a different interpretation of upgrading. Previously we have assumed that upgrading does not pose any security risk. This is based on the assumption that upgrading simply means transmitting information upward in security level. Another possibly useful interpretation of upgrading, that we will refer to as a *complete upgrade*, is that it is the movement of information upward in security level with the information completely removed from the low level. This is the interpretation in the example in Section **2**. When the system transitions to war mode, information that was previously sensitive is reclassified as secret and should no longer be visible below the secret level.

The information flow policies that we have been discussing throughout this document are meant to detect information flows from high-level subjects to low-level subjects. In this section, our concern is that low-level subjects might have access to high-level information as the result of the information being upgraded but not erased from the low level. So, the types of information flow policies we have been considering up to this point are not adequate to address this situation. The problem

is that the policies do not keep track of information flows occurring at the low level; instead they only identify information flows occurring from high-level subjects to low-level subjects. Consider the following example:

Suppose that the following processing occurs in a system:

- A piece of information is entered into object $\mathcal{A}$ at the unclassified level.
- The information in $\mathcal{A}$ is copied to object $\mathcal{B}$.
- An upgrade is used to reclassify the information in $\mathcal{A}$ at the secret level.

In order to determine that the upgrade was complete, we need to be able to determine that $\mathcal{B}$ contains information that was derived from $\mathcal{A}$. In an information flow analysis, all information at the same level is treated equally; no consideration is given to dependencies among objects at the same security level.

So, a policy for complete upgrade must take into account the dependencies between each object. Whenever the information in an object was derived from the information in an upgraded object, the derived object must be upgraded, too. Because of the complexity involved with maintaining the derivation history for each piece of information in the system, this is probably not a reasonable policy to attempt to apply to a general purpose computing system. However, it is quite reasonable to apply such a policy to well-defined applications that can reference information by value rather than by name.

A good example of such an application is a database[9]. For example, suppose that the example C^2 system in Section **2** stores its information in a database. If a crisis occurs in some part of the world and the information about that part of the world is to be upgraded, then we can examine each of the tuples in the database to determine whether it contains information about that part of the world. Any such tuples should be upgraded. Although it is possible that some other tuples in the database might have been derived from one of the upgraded tuples, our point is that it is more feasible to apply a complete upgrade policy to a system with well-defined structure in the data than to a system with highly unstructured data.

5.2.2 Reconfiguration

In this section we will consider applying information flow policies to systems that can dynamically reconfigure their resources. As with reclassification, we are concerned with reconfiguration in violation of TH-Guardedness. The violations we find will be included in the set $\mathcal{PRIV}$ and our overall system policy will be Conditional TH-Guardedness.

There are two common reasons for system reconfiguration: providing physical isolation and providing better service. Physical isolation is the separation of a node from the system. It can be used to protect the node from the rest of the system or vice versa. For example, if the operational mode of the system changes so that the processing being performed on a given node is highly critical (either

in the sense that it has a very high security classification or in the sense that the processing is mission critical), then it might be desirable to physically separate the node from the rest of the system. On the other hand, if the node is penetrated or if the node is in training or maintenance mode, it might be desirable to temporarily disconnect it from the rest of the system. System reconfiguration can lead to better service by allocating resources in a better way. For example, if a subject has a lot of resources allocated to it but is blocked waiting for user input, system performance can often be improved by taking resources away from the subject and allowing other subjects to use them.

Since the physical separation of a node from the rest of the system is likely to be initiated by a user and carried out through trusted subjects, there is usually little risk of illicit information flow as the direct result of the physical separation. If the node that is removed from the system is performing security critical processing for other nodes in the system, it is possible that the system resulting from the physical separation will no longer be secure. For example, if a system carries out all of its security processing on a single node and that node is removed from the system, the remaining system will no longer be secure. If this is a concern, it can be addressed by showing that each of the nodes in the system satisfies a composable security policy with respect to the manner in which the nodes are connected (see Section **6.1.1**). Then, the security of each node is independent of that of the remaining nodes. If the nodes cannot be shown to satisfy a composable security policy, then there is no guarantee that a secure system will still be secure after a node is removed.

In the case of system resources being reallocated to improve overall system service, we must consider how much control an untrusted subject has over the allocation of resources in the system. Since we are only concerned with operations in $\mathcal{PRIV}$, we must only consider situations in which high-level subjects may cause resources to be reallocated in a manner that is visible at the low-level. In [16] it is suggested that a stochastic system model be used to analyze these situations. The particular example considered there is that of a high-level subject reading from a low-level object as a low-level subject is writing the object. In order to provide meaningful values to the high-level subject, the system should not allow the low-level subject to modify the object while the high-level subject is reading it. So, to provide useful service, the system must reallocate the object to the high-level subject. This could be in conflict with an information flow policy though, since a low-level subject might be able to infer that control of the object has been taken away from it. The approach taken in [16] is to assign nonzero probabilities to each of the following events:

- The object being taken away from the low-level subject when the high-level subject attempts to read it.
- The object being taken away from the low-level subject when the high-level subject is not attempting to read it.

The general idea is that when the low-level subject loses control of the object, it cannot determine whether the high-level subject is accessing it or not. As this topic fits naturally under the heading of analyzing tradeoffs between security and denial of service we will discuss it further in the next contract deliverable.

5.2.3 Broadcast Messages

In distributed systems communication between processes on different nodes typically requires bidirectional communication. This poses a problem in MLS systems when the transmitter is at a lower level than the receiver. As the communication is similar to the sender writing a message to an object at the receiver's level and the receiver writing a reply to an object at the sender's level, it is not too surprising that such communication poses problems similar to that of downgrading. Since we discussed downgrading at length in Section **5.2.1**, we will not go into great detail in this section. The key point is that we must consider instances in which untrusted subjects can cause trusted subjects to initiate communication with a lower level subject or affect whether a trusted subject chooses to acknowledge a communication sent by a lower-level subject. Any such information flows represent potential security risks and must be analyzed seperately. Just as in Section **5.2.1**, a policy such as type enforcement can be used to both increase the security of the system and simplify the security analysis.

5.2.4 Operational Modes for a C^2 System

A change in operational mode is a change of system state that results in the system's future processing being fundamentally different than the system's previous processing. There are three classes of security concerns with change of operational mode:

a. The change of operational mode might result in the system enforcing a less prohibitive security policy. Thus, operations that did not result in downward flow of information previously, might do so as the result of the change of operational mode.

b. The change of operational mode might change the system security policy to prohibit actions that were previously admissible.

c. There might be additional security requirements on the changing of the operational mode itself.

As the first class is addressed by Condition TH-Guardedness, it will not be addressed further here. The latter two classes are system specific concerns rather than general concerns. So, rather than formalizing them in general, we will provide a worked example by formalizing them with respect to the example in Section **2**.

5.2.4.1 Requirements on Change of Mode— Although the precise number of operational modes for a C^2 system depends upon the mission of the system, it seems that the transportation system example in Section **2** is somewhat typical in that it possesses three main operational modes:

- Normal mode,
- Crisis mode, and

- Training mode.

In the available documentation about specific military C^2 systems, no distinction is made between battle mode and various alert modes. Thus, these are combined as the generic crisis mode, which might also include operation during a national disaster or a time of internal disorder. As noted in Section **3**, the concept of mode of operation is local to a given node. At a given instant there may be some nodes in each of the three modes of operation.

To address situations in which availability is paramount, we add:

- Emergency mode.

In this section we describe the modes of operation for a given node and the permissible transitions from one mode to another. We also sketch a policy for transitions from one mode to another.

When a node is operating in crisis mode, it is not necessarily the case that all processing on the node is related to the crisis. For instance, in the transportation system example, when a node goes into war mode, a team of individuals is identified to perform crisis related processing. However, the normal processing associated with the node may still be performed at the node. Some of the normal processing may even be done by processes acting on behalf of members of the team when they are performing in roles other than their roles as team members.

5.2.4.2 Transitions Between Modes — One of the main differences among the modes is the nature of the data that should be accessible at a node during a given mode of operation and the levels of classification associated with certain classes of data in each mode. In training mode much of the operational data is made inaccessible and is replaced with synthetic data used for training purposes. Most of this data is unclassified and must be inaccessible in other modes of operation. If the training data were accessible in other modes, there would be a vulnerability to the "wargames syndrome," that occurs when synthetic data is confused with real data, resulting in failure of the node to perform its mission.

Normal and crisis modes are differentiated by the fact that in crisis mode certain individuals are identified as the Crisis Management Team (CMT) for the crisis, and most data used or generated by the CMT is classified at higher levels than those at which the data would normally be classified. In the transportation system example deployment estimates and capabilities data relevant to the war, which are normally unclassified and secret, respectively, are both classified top secret in war mode, and all other data, except weather data, is upgraded from unclassified to secret. Requests for information from the CMT and orders from the CMT must be sanitized and downgraded prior to release to the outside world.

Emergency mode is distinguished from the other modes of operation because many of the system security mechanisms are bypassed to increase the availability of the system. For instance, access checks might be disabled, and the system might continue to run even though there was concern about the extent to which data is protected.

The allowable transitions are from normal mode to either crisis or training mode and from crisis mode to emergency mode. The restriction on transitions from training mode to crisis mode or emergency mode are fairly natural, since, prior to entering either of these modes, all synthetic training data must be deactivated and regular data must be activated. Then the appropriate regular data must be upgraded in security level if there is time. This two step process is precisely the sequence of transitions that is needed when going from training mode to normal mode to crisis mode. The restrictions on transitions to training mode also seem quite natural. It is unlikely that a training exercise would begin immediately after a crisis or an emergency, and it makes sense to finish the transition to normal activity before beginning any training. The restriction against transitions from training to crisis mode is based on the assumption that it will never be necessary to make a transition directly from one of these modes to the other mode.

Each of these modes has an associated maintenance mode. Thus, there is the normal mode of operation as well as a normal-m mode of operation. The only transition to and from a maintenance mode is via the corresponding regular mode. Thus a transition from normal-m to normal mode is allowed, but a transition from normal-m mode to crisis or crisis-m mode is not allowed. This is to insure that no activation or reclassification of data is overlooked because a transition is accomplished by a sequence of transitions through maintenance modes rather than directly from training or crisis mode to normal mode, or vice-versa.

It is worth noting that training-m mode is intended to be the mode in which true maintenance occurs if problems arise when the node is in training mode. It is also possible that part of the training exercise is to simulate maintenance. This is accomplished while the node is in training mode by feeding the node synthetic maintenance data. Similarly, normal and crisis modes may be simulated in training mode.

Figure 5-1 depicts the resulting mode transition graph. In training mode each of the four modes, crisis, crisis-m, normal, and normal-m can be simulated.

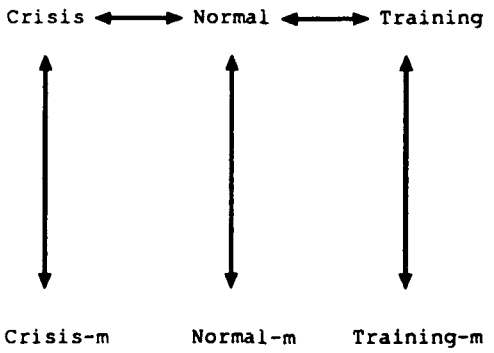


Figure 5-1. Modes

In order to formalize the policy, we define the following terminology:

- *changes_mode*(op, st) — denotes that applying op to st causes the operational mode of the system to change.

- *valid_transition*$(m1, m2)$ — denotes that it is valid to transition form mode $m1$ to mode $m2$.
- *mode*(st) — denotes the operational mode of the system when its state is st
- *authorized_transition*(op, st) — the entity initiating op is authorized to change the operational mode from *mode*(st) to *mode*(st^{op}) by issuing op in st. In our example system, we could require that both the upper command echelon and the local system administrator agree to change the operational mode before allowing the mode to change to and from war mode while leaving other transitions at the discretion of the local system administrator. Then, the definition of this function would be:
  - The operation is executed by the local system administrator and neither *mode*(st) nor *mode*(st^{op}) is war mode, otherwise
  - the operation is executed by the local administrator and the upper command echelon has approved the transition, otherwise
  - the operation is executed by the upper command echelon and the local administrator has approved the transition.

Then, the policy is that:

changes_mode(op, st)

$\Rightarrow$

valid_transition$(mode(st), mode(st^{op}))$

and *authorized_transition*(op, st)

More simply put, an operational change can be made only by authorized entities in an authorized manner.

5.2.4.3 Additional Security Requirements— In addition to requirements concerning when the operational mode may be changed, there can also be requirements concerning the processing that must occur after the mode is changed. As discussed in Section **5.2.1.2.3**, we might require that after transitioning to war mode, all of the data pertaining to the country against which war has been declared be upgraded to the secret level and no longer visible at the unclassified level. Since we have already discussed the security issues relevant to complete upgrade in Section **5.2.1.2.3**, we will not discuss them here. Instead, we merely point out here that some type of correctness analysis must be performed on the mode changing operation to ensure that it initiates processing to upgrade the information.

5.3 SUMMARY

We have now extended our notion of Strong Restrictiveness to deal with nontranquil systems and trusted operations. The resulting policy, TH-Guardedness, is too restrictive to apply to systems that allow for true downgrades and consequently is weakened to obtain Conditional TH-Guardedness. By using Conditional TH-Guardedness to analyze a system, we:

- Obtain a clear separation between the portions of the system that we know are secure and those that need to be considered further.
- Provide insight as to how the potentially insecure portions of the system might be exploited to transmit information downward in security level. This helps determine the security risk associated with the potential insecurity.

In addition to defining Conditional TH-Guardedness and using it to address security issues associated with reclassification, reconfiguration, change of operational mode, and broadcasting, we provided a worked example of a policy for change of operational mode in the context of the system described in Section 2.

Now that we have considered existing policies and adaptive policies, we will consider the relationships that hold between various policies.

SECTION 6

FURTHER COMMENTS

In this section we examine the relationships that exist between various security policies. First we consider the question of determining the policy satisfied by a composite system from the policies of each of the components. The only case in which much can be said in general is when each of the policies satisfies a *composable* security policy. After considering combining policies, we discuss the relative strengths of various policies.

6.1 CONCATENATION OF POLICIES

In the most general case, the nodes of a distributed system can be heterogeneous. In other words, it is possible that each node might be running a different operating system and have different types of communication links. The heterogeneity can also apply to the type of analysis that has been performed on the node; some nodes might have been analyzed with respect to an access control policy, other nodes with respect to an information flow policy, and other nodes might not have been analyzed at all.

There are two very distinct approaches that can be used to analyze the security of a distributed system. The first approach involves analyzing the distributed system as a monolithic entity, while the second approach involves analyzing each of the nodes separately and then attempts to make some statement about the security of the overall system in terms of the security of each node.

The advantages of the first approach are that:

a. It is a straightforward approach requiring little theoretical work.

b. It is generally applicable.

The disadvantage of the first approach is significant: It is a very labor intensive, brute-force approach. Due to the straightforwardness of this approach, it will not be discussed further in this report.

The advantages of the second approach are that:

a. It is less complex because it uses a divide and conquer approach.

b. It allows for the construction of secure components from other secure components.

The primary disadvantage of the second approach is that it requires theoretical work to construct a calculus of security policies. For example, given system A satisfying policy P_A and system B satisfying policy P_B, we need to be able to determine the policy satisfied by building a system using system A and system B as building blocks. Since it does not seem feasible to construct a general calculus, it does not seem feasible to develop a general theory for this approach. So, rather

than attempting to construct a general theory of concatenation of security policies, we will simply consider some interesting special cases.

6.1.1 Composable Policies

The only work dealing with the concatenation of security policies appearing in the literature is that dealing with the composability of security policies. A policy is said to be *composable* if:

> Given any two systems A and B that both satisfy the policy, if C is obtained by "hooking-up" A and B, then C satisfies the policy.

The concept of composability was first considered in [20]. The motivation for the consideration of composability was the observation that some policies that seemed to be reasonable formalizations of MLS security were not composable. This would suggest that even if the individual nodes of a system are shown to be "secure", there is no reason to believe that the system itself is "secure". The composability question is a hot research topic now as evidenced by the fact that most papers that introduce new security policies make a point of demonstrating that the proposed policy is composable.

6.1.1.1 Hooking-Up — One issue that is often glossed over in discussions of composability is that of the meaning of "hooking-up." It is clear that regardless of whether the nodes are secure, the composite system might be insecure if the nodes are not connected properly. For example, if a port for the transmission of high-level data from one node is connected to a port for the reception of low-level data on another node, then the composite system allows for the direct downgrading of information. So, regardless of how well the information is protected internal to each node, it is still generally necessary to consider the communication links. The common assumption to make is that the level at which a node receives a communication is the same as the level at which the sending node made the transmission. Note that this does not mean that messages from low-level processes to high-level processes are disallowed; it simply means that both the sending and receiving systems agree on the security level of each communication. For example, the sending of a message from a low-level subject on one node to a high-level subject on another node might be implemented by the sending node making a low-level output that is viewed by the receiving node as a low-level input that causes it to internally send a message to the appropriate high-level subject. On the other hand, it could be implemented as the sending node making a high-level output that the receiving node simply forwards to the appropriate high-level subject. All we are disallowing is the receiving node from ignoring the security level assigned to the message by the sending node.

It is also necessary to define the communication protocol used. For example, if the sender is blocked until the receiver acknowledges receipt of the message and the receiver is at a higher security level, then information can be signaled downward in level by the receiver selectively choosing whether to acknowledge messages. A low level subject can determine the actions of the receiver by observing whether the sender is blocked.

Since the main reason for considering security policies for nondeterministic systems is the interest in hooking-up components, it is not too surprising that most of the policies developed use the CSP formalism (which provides semantics for the composition of systems). Thus, most of the work dealing with composable security policies uses CSP as the formalism and exploits the implicit assumption that systems are hooked-up using the parallel composition operator in CSP.

This raises the question of what should be done if the communication protocol used is not consistent with the CSP parallel composition operator. One possible solution is to model the communication protocol as a separate system and define the composite system to be the composition of the component systems and the communication system instead of simply the composition of the component systems. Intuitively, this corresponds to using the built-in composition operators in CSP to simulate the appropriate communication protocol for the system at hand. If each of the component systems and the communication system satisfies a composable security policy, then the composite system obtained by combining them using the parallel composition operation will satisfy the policy, too.

There are two difficulties with this approach. The first is that, as discussed in Section **4.2.2.3**, CSP might not be the appropriate formalism to use. This is not a serious drawback since a similar approach could be used in other formalisms.

The more serious difficulty is that we have no longer achieved our goal of being able to determine the security of the entire system from the security of each of the nodes. We now must consider the security of each of the communication protocols between the nodes. Although this conclusion is not very surprising, it is worth documenting since it seems to have been ignored in the literature.

Another approach that could be used would be to define composability with respect to the type of communication protocol used between the nodes. As this would require a proof of the composability of the security policy with respect to each type of communication protocol used in the system, it seems like a less elegant approach. So, in the remainder of this section, we will assume that the "hook-ups" are accomplished through the parallel composition operator (with the understanding that the communication protocol might need to be modeled as a separate component of the composite system). The question as to whether a composition of system A with system B using communication protocol CP can always be modeled as the parallel composition of A, B, and R where R is a process representing CP is a topic for further research.

6.1.1.2 **Cause vs. Symptom** — We have tentatively concluded that any reasonable definition of security is composable. In other words, when a policy is discovered to not be composable, it appears to always be the case that it was not a reasonable definition of security in the first place.

For example, [17] describes two similar but distinct policies. The first policy, Correctability, is not composable, while the second policy, Forward Correctability, is composable. Using the terminology of Section **4.2.2.3**, the distinction between the two policies is that the latter requires "corrections" to be made after "pertubations" while the former does not. As the former defines some systems secure even though they must be able to predict the future activities of high-level subjects in order to determine the "corrections" to be made, it does not seem to be a reasonable definition of security.

Since it does not seem too surprising that the composition of two insecure systems can be more insecure than either of the component systems, it is not surprising that the composition of two "secure" machines can fail to be "secure" when "secure" has been defined incorrectly. Thus, it is not too surprising that correctability is not composable.

The specific reason that security policies such as correctability fail to be composable is that they assume that the system can predict future actions of subjects to determine the appropriate actions it should take in order to mask actions taken by high-level subjects. So, if we suppose that system A satisfies such a security policy and consider any sequence of inputs, we can determine some sequence of high-level outputs that would mask the actions taken by high-level subjects. If we compose system A with system B, a system satisfying the same security policy as A, some of the outputs from A can become inputs to B. In order to mask these inputs from low-level subjects on B, it might be necessary for B to make some high-level outputs. Some of these outputs can become high-level inputs to A. The problem now is that the action that A chose to make was dependent on it knowing the inputs it would receive in the future. As a result of the outputs A chose to make, it changed the inputs it later received, thus changing the outputs it needs to make in order to mask the high-level inputs. In certain cases, no matter what outputs A chooses to make, B is forced to make inputs to A that result in A's choice of outputs not masking the high-level inputs. This is nothing more than the time paradox on which science fiction writers are so fond of basing their stories. Each system can be thought of as observing the inputs it receives and going back in time to issue the appropriate outputs to mask the inputs that it received in the future. But, as a result of its tampering with the course of history, it changes its future so that the outputs it chose to make are inappropriate. It is analogous to the following example:

> Suppose that person A and person B play a game in which person A chooses to either put a prize in box 1 or in box 2. If B guesses the box into which A puts the prize, B wins the game.
>
> Person B cheats and watches A select the box to put the prize in and selects that box.
>
> A makes use of his time travel machine to go back in time and change his selection.
>
> By cheating again, B sees which box A selects, chooses that box, and wins again.

For any choice that B can make, A can make an earlier choice that would cause B's choice to be the losing choice. But this is irrelevant because B always gets to see A's choice before making his choice.

The significance of our viewpoint is that, if it is correct, it is misguided to focus on developing composable security policies. Rather, the goal should be to develop reasonable security policies. There is nothing wrong with testing whether a proposed policy is composable as part of the justification of the policy being reasonable, but we should not become blinded by the composability issue. Although we conjecture that any reasonable definition of security is composable, it is clear that a policy can be composable and yet not be a reasonable definition of security. For example, given two systems that allow high-level subjects to transmit information to low-level subjects, the composite system will allow high-level subjects to transmit information to low-level subjects. So,

the policy that requires the system to permit high-level subjects to communicate with low-level subjects is composable, yet it is certainly not a reasonable definition of security.

6.1.1.3 Dependencies Between Systems— A further complication in considering the composition of systems is the matter of constructing an "adequate" model of the composite system from "adequate" models of each of the components. In general, it seems that there is no way to construct an "adequate" model of the composition of systems solely from "adequate" models of each of the components. Consider the following example:

> Suppose a system allows a high-level subject to enter a text string which the system outputs at the low-level after encrypting it. If no high-level input has been made, then the system generates a random output.
>
> It is possible to make this system "secure" by making appropriate assumptions on how the encryption is done and how the system randomly generates outputs. We will not describe the assumptions here. Essentially, the idea is that there is no way to distinguish between low-level gibberish that is the ciphertext of high-level plaintext and low-level gibberish that the system randomly generates.
>
> Now, assume that we hook-up two such systems that have the same initial key. If the real system uses a deterministic "pseudo-random" generator instead of a true random generator, then the randomness injected into both systems is the same. Then, the two systems behave exactly the same except for actions caused by high-level inputs. Consequently, it is trivial to determine when high-level inputs occur and high-level subjects can signal to low-level subjects by choosing whether to make inputs.

This example illustrates that it is possible to have two nodes for which the analysis shows the nodes to be secure and the composite system to be secure, but the implementation of the composite system is insecure. This is basically the same problem as discussed in Section **4.2.2.2**. Due to determinism present in the implementation that was abstracted out of the model, certain actions that appear possible in the model are not really possible in the implementation.

The importance of this observation is that in addition to having composable security policies, it is necessary to have composable *system models*:

> We define a system model to be composable if whenever it is an "adequate" model of the system being considered, hooking it up with an "adequate" system model for another system results in an "adequate" system model for the composite system.

It is doubtful that one can generally obtain composable system models. When a system is analyzed in isolation, it is standard practice to abstract away certain implementation details that do not appear to be relevant due to assumptions made about the system. In the above example, the assumption is that it is not possible to distinguish between gibberish generated by the system and ciphertext. When the system is hooked-up with another copy of itself, this assumption is no longer valid since the copy of it generates the sequence of gibberish for the low-level subject. In general,

any time an implementation detail is abstracted away, it is possible to find some system in which abstracting the detail away changes the observable behavior of the composite system.

The same type of problem is present with policies based on stochastic system models. Here the question is whether the probability distributions for each of the nodes accurately reflect all the dependencies.

This suggests that in addition to considering communication protocols and whether the security policy is reasonable, it is also necessary to demonstrate that no dependencies among the components invalidate assumptions made in developing the system models.

6.1.1.4 Conclusions Concerning Composability

— Our conclusions with regard to composability are that:

a. Any reasonable definition of security will be composable if the systems are hooked-up properly. In particular, our Strong Restrictiveness is a composable security policy.

b. Although most of the work done to date assumes that systems are hooked-up properly, we feel that some analysis must be done to demonstrate this when a real system is analyzed. In particular, even if each of our nodes satisfies Strong Restrictiveness, we must show that the communication protocol among the nodes satisfies Strong Restrictiveness in order to show that the composite system satisfies Strong Restrictiveness.

c. We conjecture that before any attempt is made to demonstrate the security of a composite system, it is necessary to determine whether the model of the composite system obtained by hooking-up the models of each of the component systems is an "adequate" model of the true composite system. If the model obtained by composing the models of each component is not "adequate", then using the composability of the security policy to demonstrate the security of that model is of little value.

d. We have not yet considered the composability of our information flow policies for stochastic systems. We hope to do so for the final draft. Our suspicion is that it will be composable as long as the probability distributions for each of the components are independent.

It is very difficult to determine how feasible the composability approach is without trying it on a real system. We hope to be able to address some of the above concerns when we perform the THETA DOS analysis. Otherwise, the examination of these concerns must be left to a future study.

6.1.2 Heterogeneous Policies

As noted earlier, it does not seem feasible to develop a general calculus of security policies. A topic for future research is the development of a library of classes of concatenations of security policies. For example, one class in the library might represent single level workstations having a MLS node as a guard. If it is possible to provide a general argument for the security of such a system, then it would not be necessary to repeat that analysis for every such system developed.

The first step in this research would be determining which classes of systems are currently needed. The second step would be the development of security arguments for each class. The third step would be an example of using the analysis of one of the general classes to justify the security of a specific instance of the class.

6.2 RELATIONSHIPS BETWEEN POLICIES

In this section we will compare the strengths of various security policies. The goal is to demonstrate the relative strengths of various policies by comparing them with other policies.

6.2.1 Clark-Wilson vs. Type Enforcement

There is a very strong relationship between type enforcement and the Clark-Wilson model. The Clark-Wilson model described in Section **4.1.3** is formalized by:

- Defining the set of CDIs (constrained data items) to be the set of data objects that are to be protected.
- Defining the set of TPs (transformation procedures) that can be used to manipulate the CDIs. The TPs are just the well formed transactions.
- Defining the set of IVPs (integrity verification procedures) that can be used to check the integrity of the CDIs.
- Requiring the following rules be enforced:

  C1 The IVPs correctly determine the integrity of the CDIs.

  C2 The system must maintain a list of which TPs may operate on each CDI.

  E1 Each TP that may operate on a CDI must preserve the integrity of the CDI.

  E2 For each user, the system must maintain a list of which CDIs each TP may reference when operating on that user's behalf.

  C3 The list for each user of CDIs that each TP may access when operating on behalf of that user must be consistent with any separation of duty requirements.

  E3 The system must authenticate the identity of each user before allowing the user to execute a TP.

  C4 All TPs must record their actions in a system log.

  C5 Any TP that creates a CDI must ensure that the new CDI satisfies its integrity requirements.

E4 The certifier of a TP may not execute the TP. The enforcement rules of the system can only be altered by a certifier.

C1-C5 are certification procedures that are done by an individual external to the system while E1-E4 are rules enforced by the system itself.

If we define a separate domain for each TP and a type for each CDI, then E1 reduces to defining which domains may access which types. In the special case in which E2 only restricts the TPs available to each user (rather than making use of the capability of being able to also consider the CDI being accessed), E2 reduces to defining the set of domains in which each user is permitted to operate. So, although a Clark-Wilson model provides a finer degree of control to be specified, it can be built on top of a type enforcement policy in a very natural way.

6.2.2 Information Flow vs. BLP

Except for certain pathological examples, information flow policies are stronger than the Bell-LaPadula policy (see Section **4.1.2**). The intuitive reason is that if the Bell-LaPadula policy does not hold in a system, then some subject has write access to an object at a lower level or has read access to an object at a higher level. In the first case, information can be transferred downward in level as long as a low-level subject can read the object; while in the second case, information can be transferred downward in level as long as some high-level subject can write the object. Hence, the information flow policy does not hold for the system.

The pathological examples are those in which a low-level object is unreadable or a high-level object is unwritable. In the first case, even though a high-level subject might be able to write to the object, there is no information flow since no low-level subject can read the object; while in the second case, even though a low-level subject might be able to read the object, there is no information flow since no high-level subject can write the object. When an information flow policy is combined with a nonobservability policy (see Section **4.2.5**), it is no longer necessary to worry about a low-level subject being able to observe a high-level object and the only pathological case remaining is that in which a high-level subject can alter an unreadable low-level object. In this case, it is not clear that there is any security violation; it is analogous to the question of whether a tree falling in a forest makes any noise if there is no one there to hear it.

Thus, the use of a noninterference policy in conjunction with a nonobservability policy is almost always sufficient to ensure that the BLP policy holds.

As explained in Section **4.2**, it is possible, in fact typical, for a system to satisfy the Bell-LaPadula policy even though it does not satisfy an information flow policy. As evidence of how common it is for a system that satisfies the Bell-LaPadula policy to allow for information to be transmitted downward in security level, we note that the phrase *covert channel* was defined with the sole purpose of denoting such a communication channel.

6.3 SUMMARY

In this section we have discussed the matter of determining the security of a composite system by examining the security of each of its component systems. The most well-understood instance of this problem is that of composable security policies. We conjecture that matters are not quite as simple as they are made to sound in the literature and that it is doubtful that a system can be shown to be secure solely by examining the security of each of its components. This does not mean that examining the security of each of the components is not useful, it simply means that some additional work is required. In particular, the communication protocol must be examined and the model of the composite system must be adequate.

We have also considered some of the relationships between various policies. The importance of these relationships is that they illustrate the relative strength of each policy. For example, since noninterference supplemented with nonobservability almost always subsumes Bell-LaPadula, it usually places a stronger requirement on systems. Another use of the relationships is that they can aid in determining the amount of work required to enforce a policy on a system. For example, since type enforcement is very useful in implementing the Clark-Wilson enforcement rules, it might be more feasible to implement the Clark-Wilson enforcement rules on a machine with a type enforcement mechanism than on a machine that does not have a type enforcement mechanism.

SECTION 7

CONCLUSION

In this report we have provided a comprehensive discussion of existing security policies for distributed C^2 systems. We have discussed how as flaws have been discovered in security policies, new policies have been developed to address the flaws. We have pointed out flaws that we believe still exist in security policies currently being proposed for use, and suggested ways to address these flaws. In particular, we feel that our Strong Restrictiveness policy addresses a rather serious flaw in existing policies for nondeterministic systems and that our information flow policy for stochastic systems can be useful for some systems. These policies, as well as the existing policies, are too restrictive for real-world systems that need to provide capabilities such as reclassification of data, reconfiguration of resources, change of operational mode, and broadcasting. We have weakened Strong Restrictiveness to obtain Conditional TH-Guardedness to address such systems. In this sense, Conditional TH-Guardedness represents an adaptive security policy. Using Conditional TH-Guardedness, we believe that it is possible to identify the portions of a system that could lead to the disclosure of sensitive information. Once identified, application specific techniques must be used to determine the security risk present.

In the course of doing this research, we identified the following topics for further research:

- Further research needs to be done to determine whether the difficulties in stating information flow policies for nondeterministic systems are really caused by the formalisms being used or whether they are inherent problems that arise regardless of the formalism selected. If it is possible to identify a formalism that simplifies the statement of information flow policies for nondeterministic systems, then it might be possible to simplify the actual security analysis task and more easily satisfy system accreditors that the analysis is meaningful.
- With regard to the composability question, further research needs to be done to determine whether it is always possible to represent the composition of systems under a given communication protocol as the parallel composition of the systems and a process modeling the communication protocol. If it is always possible, then the assumption made in the literature that systems are composed using the parallel composition operator is partially justified. In either case, it is still important to recognize that a security analysis must be conducted on the communication protocol itself.
- Further research needs to be done concerning the types of assumptions that are typically made while performing a security analysis and whether these assumptions are violated when machines are "hooked-up." This is of particular importance with regard to nondeterministic and stochastic system models in which the behavior of the actual composite system might be significantly different than the behavior of the model composite system.
- Further research needs to be done concerning self-evolving systems. Other than inputs made by users, computer systems are typically self-evolving. The system is usually responsible for such things as scheduling the next process and obtaining the next instruction from the

executable object of the current process. Consequently, a model that ignores these aspects is inaccurate, and any analysis done with respect to such a model is potentially deficient.

- The question of applying an information flow policy to a stochastic system model needs more investigation. In addition to determining the feasibility of such an analysis, we also need to determine its worth. Covert channels that become apparent when a nondeterministic system model is refined to a stochastic system model are noisy covert channels. If the noise level is great enough, then although the channel is dangerous in theory, it might not be of concern in practice. Some potential areas of research are:

  a. Reformulating our stochastic information flow policy to require that high-level inputs cannot change the probabilities of low-level events by very much instead of requiring that the probabilities not be changed at all.

  b. Determine the magnitude of change in probabilities that a high-level subject needs to be able to make in order to have an exploitable covert channel.

  c. Determine the accuracy to which we can determine the probabilities for a real system.

  d. Investigate whether it is possible to define an information flow policy for nondeterministic system models such that whenever a system is secure with respect to that policy, there is some refinement of the system model to a stochastic system model such that the stochastic system model satisfies the stochastic information flow policy (or some variant of it).

References—

[1] Lock: Selected papers, 1985-1988. SECURE COMPUTING TECHNOLOGY CENTER, 2855 Anthony Lane South, St. Anthony, MN 55418, 1988. This is a collection of papers about the LOCK system.

[2] D.E. Bell and L.J. LaPadula. Secure computer system : Unified exposition and multics interpretation. Technical Report MTR-2997, July 1975.

[3] K. Biba. Integrity considerations for secure computer systems. Technical Report TR-3153, Mitre, April 1977.

[4] W.E. Boebert and R.Y. Kain. A practical guide to hierarchical integrity policies. In *Proceedings of the 8th National Computer Security Conference*, pages 18–27, October 1985.

[5] David F.C. Brewer and Michael J. Nash. "The Chinese Wall Security Policy". In *Proceedings of the 1989 Symposium on Security and Privacy*, pages 206–214, 1989.

[6] D.D. Clark and D.R. Wilson. A comparison of commercial and military computer security policies. In *IEEE Symposium on Computer Security and Privacy*, pages 184–194. IEEE, 1987.

[7] Dorothy E. Denning. A lattice model of secure information flow. *Communications of the ACM*, pages 236–243, May 1976.

[8] D.D. Downs, J.R. Rub, K.C.Kung, and C.S. Jordan. Issues in discretionary access control. In *IEEE Symposium on Computer Security and Privacy*, pages 208–212. IEEE, 1985.

[9] J.T. Haigh *et al.* The ldv approach to database security. In David L. Spooner and Carl. Landwehr, editors, *Database Security III: Status and Prospects*. North Holland, 1990.

[10] Todd Fine *et al.* The lock covert channel analysis report. Technical report, Secure Computing Technology Corporation, Arden Hills, MN, 1990.

[11] Wim H. Hesselink *et al.* Modalities of nondeterminacy. In W.H.J. Feijen, A.J.M. van Gasteren, D. Gries, and J. Misra, editors, *Beauty is our Business, A Birthday Salute to Edsger W. Dijkstra*. Springer-Verlag, 1990.

[12] Morrie Gasser. *Building a Secure Computer System*. Van Nostrand Reinhold Company, New York, first edition, 1988.

[13] Li Gong. A secure identity-based capability system. In *IEEE Symposium on Computer Security and Privacy*, pages 56–63. IEEE, 1989.

[14] J.T. Haigh, R. Kemmerer, J. McHugh, and W.D. Young. An experience using two covert channel analysis techinques on a real system design. *IEEE Transactions on Software Engineering*, pages 157–168, February 1987.

[15] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice-Hall International, 1985.

[16] James W. Gray, III. Probabilistic interference. In *1990 IEEE Computer Society Symposium on Research in Security and Privacy*, pages 170–179, 1990.

[17] D.M. Johnson and F.J. Thayer. Security and the composition of machines. *Proceedings IEEE/MITRE of the Workshop on the Foundations of Computer Security*, pages 72–89, 1988.

[18] R. Kain and L. Landwehr. On access checking in capability-based systems. In *IEEE Transactions on Software Engineering*, pages 202–207, February 1987.

[19] B.W. Lampson. "Protection". *Operating Systems Review*, pages 18–24, 1974.

[20] Daryl McCullough. Specifications for multi-level security and a hook-up property. *Proceedings of the 1987 Symposium on Security and Privacy*, pages 161–166, April 1987.

[21] Daryl McCullough. Noninterference and the composability of security properties. *1988 IEEE Symposium on Security and Privacy*, pages 177–186, April 1988.

[22] John McHugh, Robert L. Akers, and Millar C. Taylor. Gve users manual: The gypsy information flow tool, a covert channel analysis tool. Technical report, Computational Logic, Incorporated, July 1989.

[23] M.D. Abrams and K.W. Eggers and L.J. LaPadula and I.W. Olson. A generalized framework for access control: An informal description. In *The 13th National Computer Security Conference*, pages 135–143, 1990.

[24] J.K. Millen and C.M. Cerniglia. Computer security models. Technical Report 9531, Mitre, September 1984.

[25] Department of Defense. Trusted computer systems evaluation criteria, December 1985.

[26] Department of Defense. Military airlift command's global decision support system (gdss). Technical report, Command-In-Chief Transportation Command, June 1989.

[27] John Rushby. Mathematical foundations of the mls tool for revised special. Draft internal note, Comp. Sci. Lab, SRI International, Menlo Park, California, May 1984.

[28] John Rushby. Security policies for distributed systems: A model and calculus. Unpublished Draft, Dec. 1989.

[29] D.J. Thomsen. "Role-Based Application Design and Enforcement". In *Proceedings of the 1990 IFIP Working Group 11.3 in Database Security*, September 1990.

[30] Todd Fine. Constructively using noninterference to analyze systems. In *1990 IEEE Computer Society Symposium on Research in Security and Privacy*, pages 162–169, 1990.

[31] Todd Fine and J. Thomas Haigh and Richard C. Obrien and Dana L. Toups. Noninterference and unwinding for lock. In *The Computer Security Foundations Workshop II*, pages 22–28, 1989.

[32] James G. Williams. Modeling nondisclosure in terms of the subject-instruction stream. Submitted for public release, October 1990.

[33] Simon Wiseman. A secure capability computer system. In *IEEE Symposium on Computer Security and Privacy*, pages 86–94. IEEE, 1986.

MISSION

OF

ROME LABORATORY

Rome Laboratory plans and executes an interdisciplinary program in research, development, test, and technology transition in support of Air Force Command, Control, Communications and Intelligence (C^3I) activities for all Air Force platforms. It also executes selected acquisition programs in several areas of expertise. Technical and engineering support within areas of competence is provided to ESD Program Offices (POs) and other ESD elements to perform effective acquisition of C^3I systems. In addition, Rome Laboratory's technology supports other AFSC Product Divisions, the Air Force user community, and other DOD and non-DOD agencies. Rome Laboratory maintains technical competence and research programs in areas including, but not limited to, communications, command and control, battle management, intelligence information processing, computational sciences and software producibility, wide area surveillance/sensors, signal processing, solid state sciences, photonics, electromagnetic technology, superconductivity, and electronic reliability/maintainability and testability.